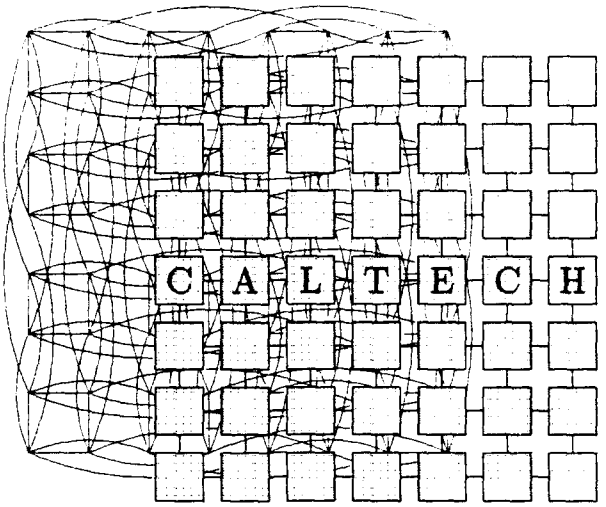


SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

FINAL TECHNICAL REPORT

This document, assembled from the semi-annual technical reports delivered to ARPA on the dates indicated within, is the final technical report for the research sponsored by the Advanced Research Projects Agency, **ARPA Order number 6202,** and monitored by the Office of Naval Research under **contract number N00014-87-K-0745.**

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-90-14

1 October 1990

Reporting Period: 16 March 1990 – 30 September 1990

Principal Investigator: Charles L. Seitz

Faculty Investigators: K. Mani Chandy
Alain J. Martin
Charles L. Seitz
Stephen Taylor
Jan van de Snepscheut

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of research activities and results for the six-and-one-half-month period, 16 March 1990 to 30 September 1990, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- Mosaic C (section 2.1).
- Mosaic programming system (section 3.1).
- The Page Kernel demonstated (section 3.3).
- Self-timed designs (section 4.1–4.8).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su

Our previous semiannual technical report includes a detailed description of the development of the Mosaic C, an experimental *fine-grain multicomputer* based on single-chip nodes and a reactive-process programming model.

Our previous report occurred just before the MOSIS 1.2μm SCMOS run that closed on 20 March 1990. Fast turnaround has allowed us to complete 2.5 iterations of design, fabrication, and testing of the Mosaic silicon in this six-and-one-half-month period.

The Mosaic project has proceeded in accordance with or faster than the schedule outlined in the previous report.

Mosaic C dRAM

A 64KB ($32K \times 16$) Mosaic C dRAM operated correctly on first silicon, and over an exceptionally wide range of operating conditions. The only anomaly discovered in testing this 1T dRAM was one-to-zero errors in several locations in the outside columns. These errors were traced to negative charge injected into the substrate by input-protection structures on pads located several hundred μm away. The input-protection structures were functioning correctly; ground bounce was causing the low input to appear as a voltage less than ground, and correcting the ground bounce in the test fixture cured the problem. The input-protection structures were replaced with an annular design that will collect the negative charges with the structure, and a guard structure was added to the outside columns of the dRAM.

This chip was also tested with a variety of deliberate disturbances, including light, alpha particles, and wide power-supply variations. The speed is right on the design target: 11MHz/V, *eg,* 44MHz at 4V operation.

The second silicon of the dRAM behaved in the same way as the first except for its susceptability to substrate charge. The yield, however, was significantly lower, but we have reason to believe that this was due to the run rather than to the changes in the design.

Memoryless Mosaic 2.1

MM2.1 is a 1.2μ version of the MM2.0 with a minor microcode change. It uses the original 3D, 4-bit-wide, synchronous router. Chips returned from fabrication in late April 1990, and are completely functional with a yield of 48/50. They have been exercised extensively in our first generation of program-development boards.

A prototype board was also made, consisting of MM2.1, our new 64KB, 1T dRAM, and two 15ns off-the-shelf EPROMs, to verify that there were no oversights

in the design of the memory interface. The setup is functional up to 27MHz at 4V, limited by the EPROM timing.

Memoryless Mosaic 3.0

MM3.0 is our first attempt at incorporating the 2D, 8-bit-wide asynchronous router into the Mosaic. The MM3 chip is assembled from the same processor as MM2.1, a version of the FMRC2.3 mesh-routing chip with several modifications (such as a 7-bit rather than a 6-bit field in the header flit to represent the relative distance), and an almost completely redesigned packet interface.

The new packet interface had to deal with a different message format – 2D *vs* 3D routing; a different protocol at the router interface – 8-bit, 2-cycle asynchronous *vs* 4-bit synchronous; and a higher data-rate at the router interface – 80MB/s *vs* 20MB/s. The scope of the required changes called for a new design rather than numerous local patches. Only the interrupt and bus-arbitration logic remained unchanged in the packet interface from the MM2.1 version. The packet interface amounts to about 30% of the active area of the MM3.0.

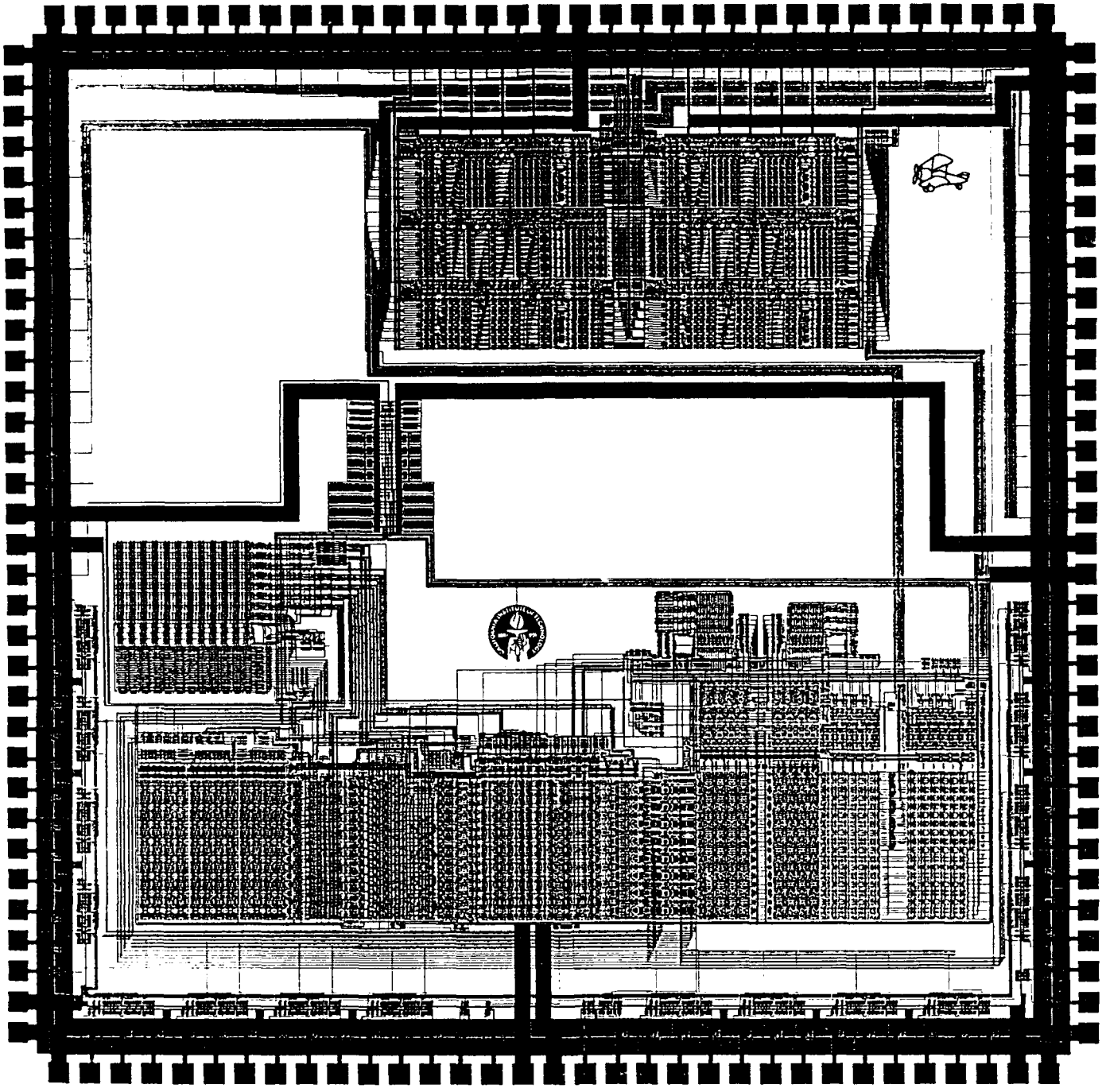
We received the chips in mid-August, and have been testing them extensively on our second-generation program-development boards. Two minor design errors were discovered during the testing. The first error was the result of an oversight in the optimization of a special case in the arbitration for storage. After this error was discovered, 12 otherwise functional chips were sent to HP to have this bug eliminated by cutting one second-metal wire with a laser. This repair was 100% successful, and allowed us to look for deeper troubles. The second design error was causing some 1 bits of packets to be received as 0s. The problem was eventually traced to the lack of a sufficient timing margin between the request and data lines at the interface between the router and the packet interface.

Both errors were fixed, and the MM3.1 was submitted for fabrication in mid-September 1990. The chips are expected in the beginning of November. Since we believe that the MM3.1 will be fully functional, we have already started the final phase of assembling the full Mosaic element, and will have it ready by the time MM3.1 chips are back.

Yield Observations

The yield for the MM3.0 was 38/50, much lower than the usual 45/50 to 48/50. The yield for the memory on this same run was 16/50 rather than the previously observed yield of 22/50.

We have tried to localize every fault to make sure that the fault is caused by fabrication rather than a marginal design. After extensive testing and numerous hours of observing chips under the microscope, 8 of the 12 bad chips have been positively identified as containing fabrication errors, and the other 4 contain probable but invisible fabrication errors.



Plot of the Memoryless Mosaic 3.1

Program-Development/Host-Interface Boards

Mosaic Processor Development Board R1.0: In order to allow meaningful software development and more comprehensive testing of the Mosaic processor, we designed a double-height (6U) VME board that holds 4 Mosaic processors connected in a two-by-two array. The board contains 128 Kbytes of SRAM per processor, and the SRAM is shared with the Sun 3/260 host by cycle stealing. The board and the processors were shown to be operating correctly to a processor clock frequency of 20 MHz — the maximum speed achievable with the 25ns external SRAM. We have fabricated ten boards and populated six of them. One of the six is used as a showpiece, and the other five are installed in various Sun 3/260s around the department. The ability to run realistic programs allowed us to detect several logic errors in the Mosaic processor that would otherwise have been missed.

Mosaic Development/Interface Board R2.0 Our decision to switch from a 3-D synchronous message network to a 2-D asynchronous network made it necessary to design a new development board. We have also taken the opportunity to modify the processor-memory and processor-VME interface to increase the clock speed achievable using our existing stock of 25ns SRAM chips. By putting the tri-state buffers needed in R1.0 on the CPU chip itself, we have also halved the total number of IC chips needed, thus making room available for installing external connectors to bring out the uncommiteed channels of the four MM3 chips. The R2.0 can thus be used as a host interface for the Mosaic multicomputer and as part of a test structure during the manufacturing of the multicomputer modules.

The 25ns SRAM allows the board and the processor to run reliably at a speed of 30 MHz. With a set of 15ns SRAM, we were able to run one of the MM3 chips at 35 MHz. The development board allowed us to discover and study a few problems with the router-processor interface. It also allowed us to discover a logic error in the condition-code register — an error that is manifested during context switching — that would never have been discovered under normal testing procedures.

Mosaic C Compiler

We have customized the Gnu C Compiler (gcc) kit to produce Mosaic assembly language code and a new assembler to produce Mosaic machine code to support the development of a compiled and dynamically-linked run time environment. As the authors of gcc claimed, the target for gcc is a CPU with 32 bit integers. For the 16 bit Mosaic, the compiler produces sub-optimal codes. We are in the process of refining the compiler to produce better code. We also need a new assembler to support the dynamic linking of object codes and to handle a set of compiler-generated directives.

Current Activities

With all of the silicon parts now tested, we are assembling the full Mosaic C node, a chip that will be approximately 9mm×10mm in 1.2μm MOSIS SCMOS. Much of the effort is in developing the built-in-test code rather than in assembling the geometry.

Negotiations with HP have been completed for the chip fabrication and packaging development for a first run of three 8×8 Mosaic boards.

A complete report on the packaging, manufacturing, testing, and early use of the Mosaic C is anticipated for the next semiannual technical report.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

Our Caltech project continues to work closely with the DARPA-supported Touchstone project at Intel Scientific Computers. The principal research activities in this period were concerned with mesh-routing chips for the Delta prototype (see section 4.8).

The project currently operates the following multicomputers: 8-node and 64-node Cosmic Cubes, a 128-node Intel iPSC/1, a 16-node Intel iPSC/2, and 32-node and 192-node Symult S2010 systems. The 192-node S2010 system is, of course, the preferred machine for users, and is accessed through the Caltech Concurrent Supercomputer Facilities. Utilization has been at a level of approximately 90% of the available node-hours.

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

3. Concurrent Computation

3.1 Fine-Grain-Multicomputer Programming Systems

Nanette J. Boden, Chuck Seitz

Significant advances in several areas of fine-grain multicomputer software have been made during the past six months.

Removal of Undue Restrictions

We are continuing our investigations into approaches that remove perceived "restrictions" on fine-grain multicomputer programming methods and on program execution that, uncorrected, could limit the application space of these machines. In previous reports we commented upon the apparent difficulty of permitting message discretion and functions in programs without perhaps introducing violation of the guarantee of message delivery. The argument is as follows: When a process is waiting for the arrival of a particular message, messages received during the interim must be buffered. Since the resources available on a node for this process are quite limited, physical space may not be available that will allow the awaited message to be received. Because we believed that unwanted messages could not be buffered within the constraints of our reactive programming model, we suggested in the last report that the need for the programming abstraction of message discretion justified an engineering solution. During the last six months we discovered a queueing formulation that successfully buffers unwanted messages while using only reactive semantics and our process-creation mechanism. This solution to the queueing problem enables a fine-grain multicomputer node to selectively receive messages *without danger of overflowing its small receive queue.* Thus, we have implemented an extremely convenient programming mechanism that uses only the reactive semantics that are ideally suited for fine-grain machines.

Fine-grain programming is clearly facilitated by the addition of a selective receive mechanism. Many functional programs can be directly translated into fine-grain programs — each function call results in the creation of a new process that eventually responds with the function value. In addition, the selective receive mechanism can be used to remove some of the simplifying assumptions that were made in early runtime systems [Oct 1989 report]. In these runtime systems, process creation was greatly simplified by assuming that if an available reference value exists for the creation of a new process on a remote node, then enough resources exist on that node for the new process. We also assumed that the code for each process resides on each node. The selective receive mechanism can be used to remove each of these restrictions. During process creation, the selective receive mechanism enables a node to wait indefinitely for a reference value to be returned by the physical node that is the eventual host for the new process. If the required code for a process is not available on a particular node, the node can use the selective receive mechanism to postpone processing of messages until the code has been dynamically linked.

Runtime System Development

Since a major goal of the Mosaic project is to provide the user with completely automatic resource management, the most recent runtime systems have focused on exploring different algorithms for process placement, code distribution, node local memory management, remote node memory management, etc. These systems incorporate much of the fundamental elements of the Cantor runtime system that was developed for the Mosaic and briefly described in our April 1989 report. In contrast to the Cantor runtime system, however, the Mosaic runtime systems have been designed so that memory and other resource demands are distributed throughout the multicomputer's available nodes. If the demands on a single node's resources threaten to overflow the available resource, the node can forward the requests or can free some of its own resources by *exporting* data structures. A design goal of this family of runtime systems is that a computation should not fail due to lack of resource until a very high percentage of the physical resources of the entire machine is actually unavailable.

Two runtime systems with different approaches to node local memory management have been developed and written in C. Using existing multicomputer nodes simulating the behavior of a Mosaic node, a Mosaic ensemble simulator has been used to partially debug these runtime systems. Further debugging, analysis, and experimentation will be performed using the Mosaic Software development boards, pending completion of a Mosaic C compiler.

Experimental Programming Notation

Since evaluation of the various runtime algorithm choices depends heavily on the original coding of the user program and on the capability of the compilers, we have been experimenting with a new notation for expressing fine-grain multicomputer programs. Although use of the fine-grain language Cantor provided much insight into the nature of fine-grain programming, the complex compiler and intermediate code of Cantor do not facilitate experimentation with such issues of interest as compiler-assisted resource management. Consequently, we are developing a C-based notation that segments a program into a collection of *definitions* that encapsulate information concerning processes and a collection of *C functions* that express conventional code. The definitions are initially written in a C language extension; a simple compiler extracts information about the processes that may be helpful in process placement and other resource management tasks, and then converts the definition code to conventional C. The C functions and the converted definitions are then compiled together using a conventional C compiler. The purpose of this effort is not to develop another fine-grain programming language, but rather to facilitate experimentation with the compilation and runtime levels of computation. This experimental programming notation is still in the design phase.

3.2 A Pascal Compiler for the Mosaic

Jan L.A. van de Snepscheut, Johan J. Lukkien

We have implemented a Pascal compiler for the Mosaic. The compiler takes a Pascal source and generates code for a single Mosaic chip. The language Pascal has been extended with primitives (derived from CSP) to support the execution of multiple processes on one processor. Communication can be performed between pairs of processes, either on the same processor or on different processors, and is synchronous. In this way we avoid the assignment of bufferspace to communicating processes.

The compiler has been used since the first Mosaics became available. A monitor program, running on a Sun workstation, loads the code into the Mosaics and communicates with the Mosaics in order to implement basic input/output functions (file I/O, terminal I/O). We have used this system to perform some fluid-flow computations.

3.3 The Page Kernel

Craig S. Steele, Chuck Seitz

The previously-described "page kernel" (PK) concurrent programming environment is now operating on the Symult S2010 multicomputer, and several test and example programs have demonstrated the functionality of its concepts. The PK is an evolution of the now-familiar reactive programming model which uses the virtual-memory capabilities of second-generation multicomputers to implement data-sharing mechanisms supporting multiple overlapping address spaces. The programmer accesses shared data structures much as in a shared-memory machine, but without the need for explicit locking to control the problems of concurrent access. The execution of the light-weight reactive processes, called *actions*, implicitly induces atomicity and consistency of data modifications. Poorly coded programs will generally run correctly but with limited effective concurrency; efficiency is improved by eliminating unnecessarily broad data consistency conditions, which may result from naive use of shared data structures. A program formulation that avoids indiscriminate writing to widely-shared data structures maximizes realizable concurrency under the PK.

While performance optimization may require careful design, many details of multicomputer programming are considerably simplified in the PK environment. Message transmission becomes implicit, as does mutual exclusion of concurrent writers to a single datum. The placement of actions and data on multicomputer nodes is handled transparently by the kernel. The physical configuration of the multicomputer hardware is hidden from the programmer; the programmer's only essential concern is to avoid reducing the problem's logical concurrency, as expressed in the program, beneath the physical concurrency, as provided by the available hardware resources. A simple triggering scheme allows actions to be scheduled

when associated data structures are changed. Actions are coded in C++, allowing definition of libraries of sharable data types of general utility, such as queue classes.

While both the kernel and the program suite are still under development, preliminary results have demonstrated near-linear speedup for problems with dozens of nodes, hundreds of actions, and thousands of shared data structures.

3.4 Multicomputer C

Marcel van der Goot, Alain Martin

Multicomputer C is a C-based concurrent programming language for message-passing multicomputers. A program consists of concurrent processes connected by channels, and communication and synchronization are done with CSP-like communication actions via the channels. During the past half-year we have worked on a manual and on a revision of the compiler. The manual describes the language design and gives implementations of the new language constructions (like communication actions). It also outlines techniques or alternatives for mapping processes on machine nodes, using time-outs in the selection of communication actions, prioritizing processes, sharing data, handling interrupts, and implementing I/O.

One change was made to the language, with the introduction of multi-sender channels. Such channels are useful to collect results computed by multiple processes in a central point, and they can often be used as an alternative to shared variables.

The compiler was reviewed and updated to allow for better diagnostics and, in particular, to facilitate code generation for a wider range of machines. The original compiler generated ANSI C for a SUN workstation. The new version is able to deal with some machine dependencies in the generated C (useful because for many machines no C compilers that implement the full standard are available), and can generate code for true multicomputers. Generating code for multicomputers involves additional difficulties when not all processors are identical, as different code may be required for different processors. Multicomputers also require a special effort to implement a global namespace for functions and processes; this can only be done at link-time. We expect the compiler to be running by the end of October, generating code for SUN workstations and for multicomputers running CE/RK. Adaptation to other medium-grain multicomputers should be straightforward.

3.5 A Concurrent Wire-Routing Program

Su-Lin Wu, Chuck Seitz

We are attempting to use multicomputers to generate wire routings of circuit boards and VLSI chips. To produce nearly optimal routes, the program will use a cost function based on physical considerations, and will also allow interaction with the user.

We have adapted the Lee-Moore algorithm for finding the shortest path between two points to a method of finding good (cheap) routes of n points. By taking advantage of existing electrically equivalent wires, this heuristic gives better routes than simply applying the two-point algorithm repeatedly. As with any attempt to solve an NP-complete problem, the n-point Lee-Moore algorithm has pathological cases, but wastes an acceptably small amount of wire in routing these.

There are easily exploitable concurrencies in this method. In the Lee-Moore algorithm's propagation phase, the parts that make up the expanding wavefront are independent and their activities can be computed concurrently. To disperse the wavefront rapidly to the nodes of the multicomputer, a wrap mapping is used. The nets and sub-nets that comprise the netlist may also be routed independently if they are confined to areas that do not intersect. The user may specify the order of the nets to be routed, but within that order the program will have some latitude to maximize concurrency. This is the classical manager–multiple-worker formulation in which the boss gives instructions to a manager who must then work within those constraints to divide the set of tasks among additional workers so that the work is completed in the shortest possible time.

Cost is based on the idea that there are limited resources available. The cost function adapts the value assigned to area, vias, and other structure to enforce behaviors desired by the user. The user assigns such costs to reflect the extent that allowing a wire to pass through that area will deplete the user's supplies.

4. VLSI Design

4.1 The Asynchronous-VLSI Project

Alain J. Martin, Dražen Borković, Steve Burns, Pieter Hazewindus, Tony Lee, José Tierno

As the project is entering its second phase, it may be appropriate to recapitulate its objectives and current status.

We have developed a novel design method for high-performance asynchronous VLSI systems. There are two main directions to the research: The first one is a high-level synthesis approach to the design of digital VLSI circuits. In our implementation of this idea, a circuit is first described as a concurrent computation in a high-level notation. It is then "compiled" into a circuit by semantics-preserving transformations. Consequently, the circuits obtained are correct by construction. (Typically, the object code is a network of CMOS pull-up and pull-down cells connected to a pad frame.)

The second aspect of the research is a novel approach to asynchronous design. We have now a complete design methodology that includes general techniques for both control and datapath, as well as a repertoire of basic cells that includes synchronizers and arbiters, generalized C-elements, bus controllers, registers, sequencing cells (D-elements), *etc.*

Although CAD was not originally a main objective of the project, an important CAD activity has developed in support of the rest of the research, since it has always been a (self-imposed) requirement on the project to test the proposed ideas by actual chip design. The set of CAD programs developed include tools for design (automatic compilation), analysis (simulation and critical path analysis), optimization for performance (transistor sizing), and physical layout (cell generation, placement and routing).

First Results

We now have a general method for designing asynchronous (and quasi delay-insensitive) circuits for any type of digital computation. We have demonstrated the practicality of the method on a series of actual MOSIS CMOS designs. All fabricated chips have been found correct on first silicon. The main chips designed include stacks, queues, routing automata, multiplier, distributed mutual exclusion (arbitration), special-purpose processor ($3X + 1$ engine); and culminated in two designs of a general-purpose 16-bit microprocessor running at 18 MIPS in 1.6μm CMOS. Since the design of this microprocessor included all main aspects of digital design (except arbitration, which was demonstrated in previous chips), the completion of the processor design was understood to be the completion of the first phase of this project.

The results of the first phase of the project can be summarized as follows: First, at the system design level, the design experiments (in particular the microprocessor) have demonstrated the flexibility and versatility of the high-level notation that we have developed. The conclusion we have drawn is that most high-level design issues are indeed concurrency issues that are best solved with the techniques and notation of concurrent computation. These results anticipate a unique design methodology for digital systems across an increasingly moveable hardware/software boundary.

Second, it is possible to design asynchronous circuits that are efficient both in area and speed. (At this point we believe that there is an irreducible area penalty compared to synchronous design, but it falls well within acceptable margins given the abundance of real-estate provided by modern technology.) With respect to speed efficiency, our experiment with demanding designs like the control part of the microprocessor indicate that rather sophisticated techniques have to be used in order to reduce the penalty due to asynchronous sequencing (completion trees, handshaking, *etc*) to an overhead comparable to that of clock skew. However, once this objective has been achieved (which is the case of the control part of the microprocessor), the asynchronous design can reap the benefits (when compared to synchronous design) of the flexible execution times and extensive concurrency provided by the concurrent computation approach.

Third, quasi delay-insensitive VLSI design exhibits remarkably robust behaviors. As previously reported, the microprocessor is operational across an unusually broad range of *VDD* voltages and range of temperature. Another remarkable feature of this type of asynchronous designs is that the power consumption is about an order of magnitude smaller than that of an equivalent synchronous design. This characteristic is of course very attractive for the design of future multicomputers that will require packaging a very large number of chips in a small volume, and also for battery-operated applications.

Designer-Assisted Compilation

The second phase of the project will concentrate on the system-level design, with a redesign of an improved version of the processor as the first step towards an entirely asynchronous system. However, we will focus first on improving the CAD tools, for the following reasons:

Our attitude towards automatic compilation has changed significantly during the project. Whereas we originally thought that we would soon use an automatic compiler for chip synthesis, we are now convinced that entirely automatic compilation will not produce high-performance design in the near future. We have an automatic compiler that has been operational for several years already. The compilation is "syntax-directed," *ie,* the compiler produces a standard circuit implementation for each syntactic construct of the language. The final design is improved by "peep-hole" optimizations. Coupled with a standard cell-layout-generation program, the compiler has been used for several automatic designs, *eg,*

for a torus-routing chip. Although such a compiler is an excellent tool for rapid prototyping, our first attempt at using it for the control part of the microprocessor convinced us that we will never get the performance we were aiming for if we follow the route of automatic compilation: The performance of the critical path of a chip like the microprocessor is just too sensitive to minor optimizations that an automatic compiler cannot even generate, let alone evaluate.

Our approach now is that of *designer-assisted compilation.* Each step of the synthesis method is applied automatically to produce a number of alternative designs. These different solutions are compared and the best (according to some criterion decided by the designer) is selected for the next step of the compilation. The procedure also includes backtracking. This approach relies on tools for performance evaluation and optimization.

The second generation of synthesis tools that we envision will integrate simulation, performance evaluation, and optimization (transistor sizing). The designer will be able or perhaps even required to make choices at different stages of the synthesis based on the results of the previous stage. As a first step toward such a system, we are designing a program for the synthesis of a straightline program into CMOS chips. The final program will include automatic cell synthesis, transistor sizing, placement and routing.

4.2 Testing Self-Timed Circuits

Pieter Hazewindus, Alain Martin

A self-timed circuit is described as a production rule set, implementing a handshaking expansion of a high-level program. For testing purposes, we use the single stuck-at model. For this model, an input or an output of a gate is either permanently at a high voltage (stuck-at-1) or at a low voltage (stuck-at-0). A circuit is tested by executing the handshaking expansion that it implements.

We are currently analyzing the testability of the control part of the first self-timed microprocessor. We have added the required testing circuitry. The revised circuit will be sent off for fabrication shortly.

4.3 Gallium Arsenide and Self-Timed Circuits

Alain J. Martin, José A. Tierno

The same techniques used for designing self-timed circuits in silicon can be applied to gallium arsenide (GaAs). However, the basic gates that are used in the implementation have to be carefully designed for reliability, noise immunity, power consumption, etc. A design style and a whole family of gates was developed so that they can be used in an "oblivious" manner, that is, requiring minimal concern for the electrical characteristics of the circuit.

A special set of pad drivers and receivers was designed to interface with this technology on chip and similar pads or *CMOS* circuits off chip. Work is in progress

now on two chips, one already in the fabrication queue and the second to be submitted before October 24th. The first circuit contains several different buffers, the basic synchronization structure for self-timed circuits, as well as smaller test features for gates and pad drivers and receivers. The second circuit contains a self-timed register file. These are being fabricated using Vitesse's enhancement depletion mode process.

4.4 Automatic Compilation of Straightline Handshaking Expansion

Dražen Borković, Alain J. Martin

As a first step towards the next generation of synthesis tools, we are designing a program for the synthesis of straightline program into CMOS chips. The final program will include automatic cell synthesis, transistor sizing, placement and routing.

The problem of positioning the state variable transitions for programs containing conditional branches ("IF" statements) was proven to be NP complete. An algorithm that solves the problem in $O(n^k)$ was developed, where k is the number of guarded commands in the "IF" statement, and n is the length of the longest guarded command.

A program for automatic generation of minimal production rules for straight-line handshaking expansions was developed, as well as one for the reset of the generated circuits. The program allows the designer to explore different options and backtrack in order to achieve the desired performance. It can also be coupled with number of other tools: inverter reshuffling, performance analysis, and cell-layout.

4.5 Automatic Custom Cell Generation and Layout

Tony Lee, Alain J. Martin

We have developed a program which generates CMOS magic cells for implementing a given set of production rules. The input production rules must be in disjunctive-normal form and the sizes of the transistors in the production rules may be specified. The output generated by this program can be used directly by *gladys*, our placement and routing program. Thus, we now have tools that will take an arbitrary set of production rules (provided it is in disjuctive-normal form) and generate a complete layout for it.

4.6 Self-Timed Arithmetic

Tony Lee, Alain J. Martin

Consider the simple shift-and-add method of multiplying two n-bit integers. If we ignore additions by zeros, then the number of partial-sum additions performed in the multiplication is determined by the number of ones in the binary representation of the multiplier. Furthermore, for each addition, the length of the longest carry-chain

is a function of the partial sum and the multiplicand. In general, the time involved in performing an arithmetic operation is greatly affected by the values of the input data. Nevertheless, this inherent variance in the latency of arithmetic operations is usually not exploited by simple synchronous systems that, for the sake of timing uniformity, operate under the worst-case delay assumption. Such a pessimistic assumption is not needed for asynchronous systems since they function properly regardless of the actual time it takes for them to perform a given computation.

Thus, we believe that efficient self-timed arithmetic circuits can be designed so that they can take advantage of the shorter latency for cases of favorable inputs and thereby yield better average performance than synchronous systems.

Our approach is to start with a high-level description of the arithmetic algorithm and then apply our synthesis method to transform the description into self-timed circuits. We have had encouraging results with the $3X + 1$ engine and the simple ALU used in the microprocessor. Currently, we are working on a multiplier that implements the shift-and-add algorithm. The layout of the multiplier has been completed and its functionality has been verified. We are now working on increasing its performance by using our timing analysis tools to size the transistors.

4.7 Performance Analysis of Linear Arrays of Asynchronous Processes

Steve Burns, Alain Martin

We have developed a method for determining the performance of linear arrays of repetitive asynchronous processes. The complexity of the procedure is related to the size of the single replicated process and not to the size of the collection of instantiated processes. This method is of great help in designing optimal pipeline stages for a computing engine, as well as *FIFO*s and stack stages for memory systems.

The method is an extension of the performance analysis techniques described in the last semi-annual report, at *TAU '91*, and in Steve Burns' forthcoming PhD thesis. Linear timing functions — the principle tool used to reduce the analysis of an infinite repetitive computation into the analysis of a finite structure — can also be used to reduce the analysis of the computation performed by an infinite array of processes into the analysis of a finite structure. Thus the performance of very large systems can be determined with very little computational effort.

These techniques have been used to compare the performance of several possible implementations of buffer processes. The implementations that achieve the highest performance have been cataloged for future use. The techniques have also been used to show that particular designs developed by other researchers are not optimal. We suggest changes to these designs which improve performance.

4.8 Fast Self-Timed Mesh-Routing Chips

Chuck Seitz

Two design-fabricate-test iterations of the Frontier mesh-routing chips (FMRC) for the Intel Touchstone Delta prototype were completed in this period. These FMRC2.2 and FMRC2.3 chips incorporated a number of improvements, described in our previous report, and aimed at increasing the reliability of high-speed data transfer on the channels.

The first mesh-routing chips fabricated in 1.2μm CMOS, the FMRC2.2, functioned correctly, but, due to the designer's misunderstanding of correcting for velocity saturation, the output drive was excessively asymmetrical. A small investigation of the output characteristics allowed the pad drivers to be corrected in the FMRC2.3 — this chip has been tested extensively both at Caltech and at Intel, and appears to be completely adequate for the Touchstone Delta prototype.

These same lessons about pad drivers have been incorporated into the pad frame of the Mosaic chips that are fabricated on these same runs.

The use of a 5-mil pad pitch in this 4492×4492, 132-pin, semi-standard frame, an experiment that Wes Hansford at MOSIS encouraged, has caused no problems

4.9 A Silicon Architecture for Adaptive Cut-Through Routing

Mike Pertel, Chuck Seitz

Previous theoretical studies have shown that the performance of multicomputer networks can be increased by using adaptive cut-through routing in place of oblivious techniques (see Ngai and Seitz, 1988). State-of-the-art oblivious routers, such as the FMRC routers described in the preceeding section, can route a packet between a given pair of nodes along only one path, regardless of the state of the network. Routers that can choose any of several paths exhibit greater utilization of network bandwidth, better traffic balancing, and increased fault tolerance. To implement the ideas from the earlier work, we have developed a simple architecture for performing multipath routing. The architecture confines the design space to allow detailed simulation, but does not appear to limit flexibility.

A routing algorithm for multicomputer networks must be deadlock-free to be practical. The oblivious routers avoid deadlock by using dimension-order routing; a multipath router requires another mechanism. A key idea from the theoretical studies is to avoid deadlock by misrouting. Deadlock is impossible if a router never blocks its input channels. By using any available output channel, it can rid itself of packets that it cannot buffer. The earlier studies showed that even if misrouting is used to avoid deadlock, it can be made very rare by throttling the network traffic. The architecture supports deadlock avoidance by being able to misroute a packet from any input to any output. The congestion control required to cause misrouting to be required only rarely is handled by requiring packet sources to

await an acknowledge message from the destination before further sending. This technique also assures packet-order preservation despite the existence of multiple paths between source and destination.

Once the problem of deadlock is resolved, we are free to consider any number of ways to route packets. The path of a packet in an oblivious routing network is fixed by the deadlock-avoidance scheme, but misrouting eliminates this restriction. The adaptive router can forward an incoming packet along any profitable output channel. The exact definition of what constitutes a profitable channel assignment depends on the specific routing algorithm, but in general a channel is profitable if it reduces the packet's distance from its destination. We can avoid making misrouting a special case by regarding any output assignment as profitable when input blocking becomes imminent. Other than this, the definition of a profitable assignment is left open, thus we maintain the flexibility to implement virtually any specific algorithm. Since there will generally be more than one profitable output for a packet, it is necessary to choose one assignment from multiple candidates. Moreover, output assignments must be made fairly in the sense that any packet awaiting an output eventually gets one.

An architecture to support this framework must be able to connect any input to any output for misrouting. It must also have buffering to allow packets to wait for profitable outputs to become available without blocking. This suggests a simple structure with FIFOs and a crossbar. By placing the FIFOs on the input channels, we can use the filling of the input queue to trigger misrouting. The requirement that access to output channels be fair between the inputs, and the necessity of ensuring that each input is connected to at most one output (and each output to at most one input) suggest a central decision structure. An important lesson from the theoretical studies was that simultaneous arrival of multiple new packets needing output assignments is very rare. This suggests that the hardware for reading/writing packet headers and computing/making profitable assignments can be shared by all inputs, rather than duplicated, with negligible increase in average assignment latency.

The incoming packets awaiting output assignments are serviced sequentially. This eliminates the need to duplicate logic for computing assignments, and trivializes the problem of mutual exclusion between assignments. More importantly, by serving the inputs round-robin, we guarantee fair access to the output channels. When an input is served, we compute the profitability of each output in parallel. If no profitable output is free, no assignment is made. If at least one profitable assignment is possible, then one is chosen. In the case where the profitability of an assignment is discrete (eg, binary), we arbitrarily select one of the most profitable assignments. If profitability is continuous, we merely choose the most profitable. We assume that the determination of output profitability can be done in one cycle. Based upon the theoretical studies, this is a reasonable assumption. Given that it only takes one cycle to service an input, and that simultaneous header arrivals are

rare, sequential service does not compromise efficiency, and it solves the problems associated with doing channel assignment quite cleanly.

We have developed a simulator for the architecture described. The profitability of an assignment is determined by a small C function. We are proceeding to compare the performance of several definitions of profitability under different traffic conditions to select the best alternative. The simple architecture we have described has the flexibility to implement the promising algorithms developed during the earlier theoretical studies, yet it dramatically reduces the design space to explore. In its simplicity, the architecture demonstrates that it is not difficult to design a practical adaptive router.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports

1 October 1990

Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check for a minimum of $50.00 drawn on a U.S. bank in U.S. currency, payable to CALTECH.

CS-TR-90-13	$3.50	Weakest Preconditions for Progress Lukkien, Johan J, and Jan L A van de Snepscheut
CS-TR-90-12	$2.50	Performance Analysis and Optimization of Asynchronous Circ. Produced by Martin Analysis Burns, Steven
CS-TR-90-11	$2.00	Characterizing sl NP and Measuring Instance Complexity Judd, Stephen
CS-TR-90-10	$5.00	Primer for Program Composition Notation Chandy, K Mani, and Stephen Taylor
CS-TR-90-09	$2.00	Asynchronous Circuits for Token-Ring Mutual Exclusion Martin, Alain J
CS-TR-90-07	$2.00	Compiler Optimization of Array Data Storage Gupta, Rajiv, and Jim Kajiya
CS-TR-90-06	$2.00	Distributed Sorting Hofstee, H Peter, Alain J Martin, and Jan van de Snepscheut
CS-TR-90-05	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-90-03	$3.00	Program Composition Project Chandy, K Mani, with Stephen Taylor, Carl Kesselman, and Ian Foster
CS-TR-90-02	$2.00	Limitations to Delay-Insensitivity in Asynchronous Circuits Martin, Alain J
CS-TR-90-01	$3.00	Properties of the V-C Dimension, MS Thesis Fyfe, Andrew
CS-TR-89-12	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-89-11	$9.00	Reactive-Process Programming and Distributed Discrete-Event Simulation, PhD Thesis Su, Wen-King
CS-TR-89-10	$7.00	Silicon Models of Early Audition, PhD Thesis Lazarro, John
CS-TR-89-09	$15.00	Framework for Adaptive Routing in Multicomputer Networks, PhD Thesis Ngai, John
CS-TR-89-07	$6.00	Constraint Methods for Neural Networks and Computer Graphics, PhD Thesis Platt, John
CS-TR-89-06	$1.00	First Asynchronous Microprocessor: The Test Results Martin, Alain J, Steven M Burns, T K Lee, Drazen Borkovic, and Pieter J Hazewindus
CS-TR-89-05	$2.00	Essence of Distributed Snapshots Chandy, K Mani
CS-TR-89-04	$5.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-89-03	$3.00	Feature-oriented Image Enhancement with Shock Filters, I Rudin, Leonid I with Stanley Osher

CS-TR-89-02	$3.00	Design of an Asynchronous Microprocessor Martin, Alain J
CS-TR-89-01	$4.00	Programming in VLSI From Communicating Processes to Delay-insensitive Circuits Martin, Alain J
CS-TR-88-22	$2.00	Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm Su, Wen-King, and Charles L Seitz
CS-TR-88-21	$3.00	Winner-Take-All Networks of O(N) Complexity Lazzaro, John, with S Ryckebusch, M A Mahowald, and C A Mead
CS-TR-88-20	$7.00	Neural Network Design and the Complexity of Learning Judd, J Stephen
CS-TR-88-19	$5.00	Controlling Rigid Bodies with Dynamic Constraints Barzel, Ronen
CS-TR-88-18	$3.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-88-17	$3.00	Constrained Differential Optimization for Neural Networks Platt, John C, and Alan H Barr
CS-TR-88-16	$3.00	Programming Parallel Computers Chandy, K Mani
CS-TR-88-15	$13.00	Applications of Surface Networks to Sampling Problems in Computer Graphics, PhD Thesis Von Herzen, Brian
CS-TR-88-14	$2.00	Syntax-directed Translation of Concurrent Programs into Self-timed Circuits Burns, Steven M, and Alain J Martin
CS-TR-88-13	$2.00	Message-Passing Model for Highly Concurrent Computation Martin, Alain J
CS-TR-88-12	$4.00	Comparison of Strict and Non-strict Semantics for Lists, MS Thesis Burch, Jerry R
CS-TR-88-11	$5.00	Study of Fine-Grain Programming Using Cantor, MS Thesis Boden, Nanette J
CS-TR-88-10	$3.00	Reactive Kernel, MS Thesis Seizovic, Jacov
CS-TR-88-07	$3.00	Hexagonal Resistive Network and the Circular Approximation Feinstein, David I
CS-TR-88-06	$3.00	Theorems on Computations of Distributed Systems Chandy, K Mani
CS-TR-88-05	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-88-04	$3.00	Cochlear Hydrodynamics Demystified Lyon, Richard F, and Carver A Mead
CS-TR-88-03	$4.00	PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics, MS Thesis Gupta, Rajiv
CS-TR-88-02	$4.00	Automated Compilation of Concurrent Programs into Self-timed Circuits, MS Thesis Burns, Stephen M
CS-TR-88-01	$3.00	C Programmer's Abbreviated Guide to Multicomputer Programming Seitz, Charles, Jakov Seizovic, and Wen-King Su
5258:TR:88	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5256:TR:87	$2.00	Synthesis Method for Self-timed VLSI Circuits Martin, Alain. *(current supply only: see Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design 224-229, Oct'87)*

5253:TR:88 $2.00 Synthesis of Self-Timed Circuits by Program Transformation
Burns, Steven M, and Alain J Martin

5251:TR:87 $2.00 Conditional Knowledge as a Basis for Distributed Simulation
Chandy, K Mani, and Jay Misra

5250:TR:87 $10.00 Images, Numerical Analysis of Singularities and Shock Filters, PhD Thesis
Rudin, Leonid Iakov

5249:TR:87 $6.00 Logic from Programming Language Semantics, PhD Thesis
Choo, Young-il

5247:TR:87 $6.00 VLSI Concurrent Computation for Music Synthesis, PhD Thesis
Wawrzynek, John

5246:TR:87 $3.00 Framework for Adaptive Routing
Ngai, John Y, and Charles L Seitz

5244:TR:87 $3.00 Multicomputers
Athas, William C, and Charles L Seitz

5243:TR:87 $5.00 Resource-Bounded Category and Measure in Exponential Complexity Classes, PhD Thesis
Lutz, Jack H

5242:TR:87 $8.00 Fine Grain Concurrent Computations, PhD Thesis
Athas, William C

5241:TR:87 $3.00 VLSI Mesh Routing Systems, MS Thesis
Flaig, Charles M

5240:TR:87 $2.00 Submicron Systems Architecture
ARPA Semiannual Technical Report

5239:TR:87 $3.00 Trace Theory and Systolic Computations
Rem, Martin

5238:TR:87 $7.00 Incorporating Time in the New World of Computing System, MS Thesis
Poh, Hean Lee

5236:TR:86 $4.00 Approach to Concurrent Semantics Using Complete Traces, MS Thesis
Van Horn, Kevin S

5235:TR:86 $4.00 Submicron Systems Architecture
ARPA Semiannual Technical Report

5234:TR:86 $3.00 High Performance Implementation of Prolog
Newton, Michael O

5233:TR:86 $3.00 Some Results on Kolmogorov-Chaitin Complexity, MS Thesis
Schweizer, David Lawrence

5232:TR:86 $4.00 Cantor User Report
Athas, W C, and C L Seitz

5230:TR:86 $24.00 Monte Carlo Methods for 2-D Compaction, PhD Thesis
Mosteller, R C

5229:TR:86 $4.00 anaLOG - A Functional Simulator for VLSI Neural Systems, MS Thesis
Lazzaro, John

5228:TR:86 $3.00 On Performance of k-ary n-cube Interconection Networks
Dally, Wm J

5227:TR:86 $18.00 Parallel Execution Model for Logic Programming, PhD Thesis
Li, Pey-yun Peggy

5223:TR:86 $15.00 Integrated Optical Motion Detection, PhD Thesis
Tanner, John E

5221:TR:86 $3.00 Sync Model: A Parallel Execution Method for Logic Programming
Li, Pey-yun Peggy, and Alain J Martin. *(current supply only: see Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86)*

5220:TR:86 $4.00 Submicron Systems Architecture
ARPA Semiannual Technical Report

5215:TR:86 $2.00 How to Get a Large Natural Language System into a Personal Computer
Thompson, Bozena H, and Frederick B Thompson

5214:TR:86 $2.00 ASK is Transportable in Half a Dozen Ways
Thompson, Bozena H, and Frederick B Thompson

5212:TR:86 $2.00 On Seitz' Arbiter
Martin, Alain J

5210:TR:86 $2.00 Compiling Communicating Processes into Delay-Insensitive VLSI Circuits
Martin, Alain. *(current supply only: see Distributed Computing v 1 no 4 (1986))*

5207:TR:86 $2.00 Complete and Infinite Traces: A Descriptive Model of Computing Agents
van Horn, Kevin

5205:TR:85 $2.00 Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity
Schweizer, David, and Yaser Abu-Mostafa

5204:TR:85 $3.00 An Inverse Limit Construction of a Domain of Infinite Lists
Choo, Young-Il

5202:TR:85 $15.00 Submicron Systems Architecture
ARPA Semiannual Technical Report

5200:TR:85 $18.00 ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation, PhD Thesis
Whelan, Dan

5198:TR:85 $8.00 Neural Networks, Pattern Recognition and Fingerprint Hallucination, PhD Thesis
Mjolsness, Eric

5197:TR:85 $7.00 Sequential Threshold Circuits, MS thesis
Platt, John

5195:TR:85 $3.00 New Generalization of Dekker's Algorithm for Mutual Exclusion
Martin, Alain J. *(current supply only: see Information Processing Letters 23 295-297 1986)*

5194:TR:85 $5.00 Sneptree - A Versatile Interconnection Network
Li, Pey-yun Peggy, and Alain J Martin

5193:TR:85 $2.00 Delay-insensitive Fair Arbiter
Martin, Alain J

5190:TR:85 $3.00 Concurrency Algebra and Petri Nets
Choo, Young-il

5189:TR:85 $10.00 Hierarchical Composition of VLSI Circuits, PhD Thesis
Whitney, Telle

5185:TR:85 $11.00 Combining Computation with Geometry, PhD Thesis
Lien, Sheue-Ling

5184:TR:85 $7.00 Placement of Communicating Processes on Multiprocessor Networks, MS Thesis
Steele, Craig

5179:TR:85 $3.00 Sampling Deformed, Intersecting Surfaces with Quadtrees, MS Thesis
Von Herzen, Brian P

5178:TR:85 $9.00 Submicron Systems Architecture
ARPA Semiannual Technical Report

5174:TR:85 $7.00 Balanced Cube: A Concurrent Data Structure
Dally, William J, and Charles L Seitz

5172:TR:85 $6.00 Combined Logical and Functional Programming Language
Newton, Michael

5168:TR:84 $3.00 Object Oriented Architecture
Dally, Bill, and Jim Kajiya

5165:TR:84 $4.00 Customizing One's Own Interface Using English as Primary Language
Thompson, B H, and Frederick B Thompson

5164:TR:84	$13.00	ASK French - A French Natural Language Syntax, MS Thesis Sanouillet, Remy
5160:TR:84	$7.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5158:TR:84	$6.00	VLSI Architecture for Sound Synthesis Wawrzynek, John, and Carver Mead
5157:TR:84	$15.00	Bit-Serial Reed-Solomon Decoders in VLSI, PhD Thesis Whiting, Douglas
5147:TR:84	$4.00	Networks of Machines for Distributed Recursive Computations Martin, Alain, and Jan van de Snepscheut
5143:TR:84	$5.00	General Interconnect Problem, MS Thesis Ngai, John
5140:TR:84	$5.00	Hierarchy of Graph Isomorphism Testing, MS Thesis Chen, Wen-Chi
5139:TR:84	$4.00	HEX: A Hierarchical Circuit Extractor, MS Thesis Oyang, Yen-Jen
5137:TR:84	$7.00	Dialogue Designing Dialogue System, PhD Thesis Ho, Tai-Ping
5136:TR:84	$5.00	Heterogeneous Data Base Access, PhD Thesis Papachristidis, Alex
5135:TR:84	$7.00	Toward Concurrent Arithmetic, MS Thesis Chiang, Chao-Lin
5134:TR:84	$2.00	Using Logic Programming for Compiling APL, MS Thesis Derby, Howard
5133:TR:84	$13.00	Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems, PhD Thesis Lin, Tzu-mu
5132:TR:84	$10.00	Switch Level Fault Simulation of MOS Digital Circuits, MS Thesis Schuster, Mike
5129:TR:84	$5.00	Design of the MOSAIC Processor, MS Thesis Lutz, Chris
5128:TM:84	$3.00	Linguistic Analysis of Natural Language Communication with Computers Thompson, Bozena H
5125:TR:84	$6.00	Supermesh, MS Thesis Su, Wen-King
5123:TR:84	$14.00	Mossim Simulation Engine Architecture and Design Dally, Bill
5122:TR:84	$8.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5114:TM:84	$3.00	ASK As Window to the World Thompson, Bozena, and Fred Thompson
5112:TR:83	$22.00	Parallel Machines for Computer Graphics, PhD Thesis Ulner, Michael
5106:TM:83	$1.00	Ray Tracing Parametric Patches Kajiya, James T
5104:TR:83	$9.00	Graph Model and the Embedding of MOS Circuits, MS Thesis Ng, Tak-Kwong
5094:TR:83	$2.00	Stochastic Estimation of Channel Routing Track Demand Ngai, John
5092:TM:83	$2.00	Residue Arithmetic and VLSI Chiang, Chao-Lin, and Lennart Johnsson

5091:TR:83	$2.00	Race Detection in MOS Circuits by Ternary Simulation Bryant, Randal E
5090:TR:83	$9.00	Space-Time Algorithms: Semantics and Methodology, PhD Thesis Chen, Marina Chien-mei
5089:TR:83	$10.00	Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits Lin, Tzu-Mu, and Carver A Mead
5086:TR:83	$4.00	VLSI Combinator Reduction Engine, MS Thesis Athas, William C Jr
5082:TR:83	$10.00	Hardware Support for Advanced Data Management Systems, PhD Thesis Neches, Philip
5081:TR:83	$4.00	RTsim - A Register Transfer Simulator, MS Thesis Lam, Jimmy
5074:TR:83	$10.00	Robust Sentence Analysis and Habitability Trawick, David
5073:TR:83	$12.00	Automated Performance Optimization of Custom Integrated Circuits, PhD Thesis Trimberger, Steve
5065:TR:82	$3.00	Switch Level Model and Simulator for MOS Digital Systems Bryant, Randal E
5054:TM:82	$3.00	Introducing ASK, A Simple Knowledgeable System Conf on App'l Natural Language Processing Thompson, Bozena H, and Frederick B Thompson
5051:TM:82	$2.00	Knowledgeable Contexts for User Interaction Proc Nat'l Computer Conference Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho
5035:TR:82	$9.00	Type Inference in a Declarationless, Object-Oriented Language, MS Thesis Holstege, Eric
5034:TR:82	$12.00	*Hybrid Processing*, PhD Thesis Carroll, Chris
5033:TR:82	$4.00	MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual Schuster, Mike, Randal Bryant, and Doug Whiting
5029:TM:82	$4.00	POOH User's Manual Whitney, Telle
5018:TM:82	$2.00	Filtering High Quality Text for Display on Raster Scan Devices Kajiya, Jim, and Mike Ullner
5017:TM:82	$2.00	Ray Tracing Parametric Patches Kajiya, Jim
5015:TR:82	$15.00	VLSI Computational Structures Applied to Fingerprint Image Analysis Megdal, Barry
5014:TR:82	$15.00	Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture, PhD Thesis Lang, Charles R Jr
5012:TM:82	$2.00	Switch-Level Modeling of MOS Digital Circuits Bryant, Randal
5000:TR:82	$6.00	Self-Timed Chip Set for Multiprocessor Communication, MS Thesis Whiting, Douglas
4684:TR:82	$3.00	Characterization of Deadlock Free Resource Contentions Chen, Marina, Martin Rem, and Ronald Graham
4655:TR:81	$20.00	Proc Second Caltech Conf on VLSI Seitz, Charles, ed.
4090-TR-80	$3.00	VLSI Based Real-Time Hidden Surface Elimination Display System, MS Thesis Demetrescu, Stefan G

3760:TR:80	$10.00	Tree Machine: A Highly Concurrent Computing Environment, PhD Thesis Browning, Sally
3759:TR:80	$10.00	Homogeneous Machine, PhD Thesis Locanthi, Bart
3710:TR:80	$10.00	Understanding Hierarchical Design, PhD Thesis Rowson, James
3340:TR:79	$26.00	Proc. Caltech Conference on VLSI (1979) Seitz, Charles, ed
2276:TM:78	$12.00	Language Processor and a Sample Language Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

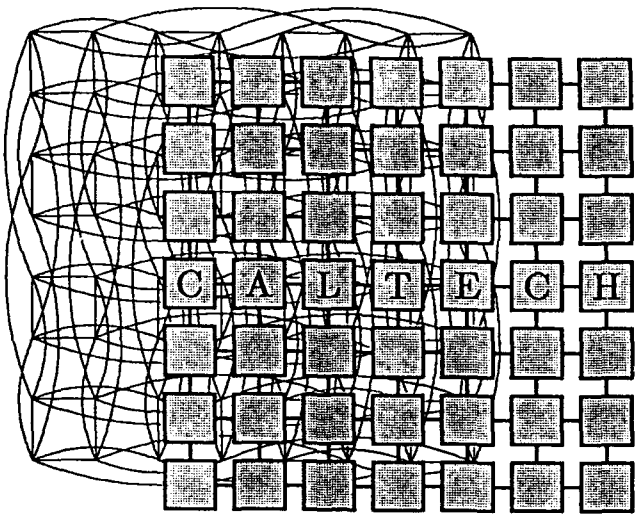
_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

___CS-TR-90-13	___CS-TR-88-19	___5244:TR:87	___5204:TR:85	___5139:TR:84	___5073:TR:83
___CS-TR-90-12	___CS-TR-88-18	___5243:TR:87	___5202:TR:85	___5137:TR:84	___5065:TR:82
___CS-TR-90-11	___CS-TR-88-17	___5242:TR:87	___5200:TR:85	___5136:TR:84	___5054:TM:82
___CS-TR-90-10	___CS-TR-88-16	___5241:TR:87	___5198:TR:85	___5135:TR:84	___5051:TM:82
___CS-TR-90-09	___CS-TR-88-15	___5240:TR:87	___5197:TR:85	___5134:TR:84	___5035:TR:82
___CS-TR-90-07	___CS-TR-88-14	___5239:TR:87	___5195:TR:85	___5133:TR:84	___5034:TR:82
___CS-TR-90-06	___CS-TR-88-13	___5238:TR:87	___5194:TR:85	___5132:TR:84	___5033:TR:82
___CS-TR-90-05	___CS-TR-88-12	___5236:TR:86	___5193:TR:85	___5129:TR:84	___5029:TM:82
___CS-TR-90-03	___CS-TR-88-11	___5235:TR:86	___5190:TR:85	___5128:TM:84	___5018:TM:82
___CS-TR-90-02	___CS-TR-88-10	___5234:TR:86	___5189:TR:85	___5125:TR:84	___5017:TM:82
___CS-TR-90-01	___CS-TR-88-07	___5233:TR:86	___5185:TR:85	___5123:TR:84	___5015:TR:82
___CS-TR-89-12	___CS-TR-88-06	___5232:TR:86	___5184:TR:85	___5122:TR:84	___5014:TR:82
___CS-TR-89-11	___CS-TR-88-05	___5230:TR:86	___5179:TR:85	___5114:TM:84	___5012:TM:82
___CS-TR-89-10	___CS-TR-88-04	___5229:TR:86	___5178:TR:85	___5112:TR:83	___5000:TR:82
___CS-TR-89-09	___CS-TR-88-03	___5228:TR:86	___5174:TR:85	___5106:TM:83	___4684:TR:82
___CS-TR-89-07	___CS-TR-88-02	___5227:TR:86	___5172:TR:85	___5104:TR:83	___4655:TR:81
___CS-TR-89-06	___CS-TR-88-01	___5223:TR:86	___5168:TR:84	___5094:TR:83	___4090-TR-80
___CS-TR-89-05	___5258:TR:88	___5221:TR:86	___5165:TR:84	___5092:TM:83	___3760:TR:80
___CS-TR-89-04	___5256:TR:87	___5220:TR:86	___5164:TR:84	___5091:TR:83	___3759:TR:80
___CS-TR-89-03	___5253:TR:88	___5215:TR:86	___5160:TR:84	___5090:TR:83	___3710:TR:80
___CS-TR-89-02	___5251:TR:87	___5214:TR:86	___5158:TR:84	___5089:TR:83	___3340:TR:79
___CS-TR-89-01	___5250:TR:87	___5212:TR:86	___5157:TR:84	___5086:TR:83	___2276:TM:78
___CS-TR-88-22	___5249:TR:87	___5210:TR:86	___5147:TR:84	___5082:TR:83	___
___CS-TR-88-21	___5247:TR:87	___5207:TR:86	___5143:TR:84	___5081:TR:83	___
___CS-TR-88-20	___5246:TR:87	___5205:TR:85	___5140:TR:84	___5074:TR:83	___



SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-90-05

15 March 1990

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202; and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-90-05

15 March 1990

Reporting Period:	1 November 1989 – 15 March 1990
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	K. Mani Chandy
	Alain J. Martin
	Charles L. Seitz
	Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a *summary* of research activities and results for the four-and-one-half-month period, 1 November 1989 to 15 March 1990, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- Mosaic is ready to build (section 2.1).
- Fully functional Memoryless Mosaic chips (section 2.1.4).
- High-density Mosaic memory (sections 2.1.2 and 4.7).
- Mosaic program-development boards (section 2.1.5).
- New message-order semantics (section 3.2).
- Cache memory for an asynchronous microprocessor (section 4.2).
- New results in transistor-sizing for asynchronous circuits (section 4.4).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su

The development of the Mosaic C, an experimental *fine-grain multicomputer* based on single-chip nodes and a reactive-process programming model, is entering its final stages. This system-building experiment incorporates much of what we have learned over the past decade about the architecture, design, and programming of multicomputers. Indeed, many of our recent contributions to the development of medium-grain multicomputers (see section 2.2), such as low-latency message-passing networks and streamlined message handling in the node operating system, have come directly out of our investigations of the design and programming of fine-grain multicomputers, in which these problems are substantially more difficult.

The Mosaic C project includes numerous interacting subtasks ranging from chip design and system packaging to programming-system development and application studies. The fabrication of a large-scale prototype is now forcing decisions on design options that have deliberately been left open; hence, we offer in this semi-annual technical report a detailed status report on the entire project.

2.1.1 Architecture rationale

The Mosaic C is a member of a class of programmable, MIMD, distributed-memory, concurrent computers called multicomputers. (See the article by Athas & Seitz in the August 1988 issue of *IEEE Computer* for background.) These machines consist of an ensemble of N programmable computers called *nodes,* each of which may support many concurrent processes. Interprocess communication takes place by messages that are conveyed and routed between nodes by a direct communication network. Multicomputers are true VLSI architectures: They can be scaled to very large numbers of nodes, and can exploit the performance and complexity of submicron-feature-size microelectronic technologies. Multicomputers have proven to possess a broad application span, and allow explicitly concurrent programs to be expressed in a variety of programming notations.

The commercial examples of multicomputers manufactured by Intel Scientific Computers, Symult Systems, and N-CUBE are based on a computational model, prototype developments, and system software developed in our research project. They are all *medium-grain multicomputers* in which configurations capable of substantially outperforming conventional vector supercomputers consist of hundreds of nodes with several MBytes of storage per node.

Shared-memory multiprocessors are not as scalable as multicomputers; however, multiprocessors can certainly be scaled into the range of hundreds of processors, and in this range possess some advantages over multicomputers. Among MIMD systems,

the exclusive "niche" of the multicomputer begins at about $N \geq 2^{10}$ nodes. We understand today how to scale multicomputers to at least $N = 2^{21}$ nodes.

Although medium-grain machines can be scaled into the range of thousands of nodes, economics dictates that multicomputers with large N will employ small nodes. Consider this constant-silicon-cost argument. A medium-grain multicomputer with $N = 256$ and 4MB/node requires about $1m^2$ of silicon in a modern $1\mu m$ CMOS process. About 60% of the $4{,}000mm^2$ silicon area of each node is devoted to the 4MB of primary memory. Suppose that the essential parameters of a multicomputer design, N and the node size, were shifted by a factor of 2^6, so that a machine would consist of 16K nodes, each with 64KB of memory. Such a machine would have the same total memory and silicon-area cost as a 256-node medium-grain multicomputer; however, because the performance of the instruction-interpreting processor is not reduced in proportion to its area, the aggregate peak performance of this fine-grain multicomputer system would be significantly higher than that of a medium-grain multicomputer. In fact, because a single node would require only about $60mm^2$ and could be integrated onto a single chip, the localization of communication between the processor and memory allows a single-chip node to exhibit performance that is comparable to that of the multi-chip node used in medium-grain systems.

The Mosaic C closely fits this description of a fine-grain multicomputer. It is based on single-chip nodes, and we are working toward assembling a prototype consisting of 16K nodes. We recognized long ago that multicomputers with single-chip nodes were technologically the most attractive point within the space of multicomputer designs. As was reported in 1985 (see Seitz's article in the January 1985 issue of the *CACM*), the Cosmic Cube was developed by our research group (in 1981–83) to study the programming techniques and applications of the multicomputer systems that we expected could be constructed with single-chip nodes by about 1991.

We expect that the Mosaic C will become the origin of a new scaling track for multicomputers. The fine-grain, single-chip-node track offers substantially higher performance and performance/cost than medium-grain multicomputers, and is centered in a niche that is beyond the scaling range of multiprocessors, while still providing the wide application span of MIMD systems.

2.1.2 The Mosaic C node

Because single-chip nodes were a stipulation of the Mosaic experiment, it is most convenient to describe this system "bottom-up," starting from the single-chip node element.

The Mosaic C node was designed and laid out using the MOSIS SCMOS scalable-CMOS design rules, and uses fully restored logic with two-phase clocking. It is typical of chips designed with these rules and disciplines to be highly tolerant of process variations. The 50C design clock rate is 40MHz at 4V in $1.2\mu m$ SCMOS, and tests

of parts fabricated in 1.6μm CMOS confirm that we will achieve this performance by a considerable margin.

The major parts were initially fabricated separately for testing and yield characterization, and are listed below:

Part	Lambda dimensions	As fabricated in 1.2um CMOS
16KB 4T dRAM	14000, 7700	8.4mm x 4.6mm = 38.6 sq mm
64KB 1T dRAM	14000, 12000	8.4mm x 7.2mm = 60.5 sq mm
8KB bootstrap ROM	7000, 3000	4.2mm x 1.8mm = 7.6 sq mm
Processor	4000, 3000	2.3mm x 1.8mm = 4.3 sq mm
Router	1500, 3000	0.9mm x 1.8mm = 1.6 sq mm
Packet Interface	1500, 3000	0.9mm x 1.8mm = 1.6 sq mm
TOTAL (16KB dRAM)	14000, 10700	8.4mm x 6.4mm = 53.8 sq mm
TOTAL (64KB dRAM)	14000, 16000	8.4mm x 9.6mm = 80.6 sq mm

These dimensions are slightly exaggerated to allow for the routing space between the parts. Allowing also for the pad frame and space to route signals to it, the chip dimensions for the version that uses the 16KB 4T dRAM will be approximately 9.0mm$\times$7.4mm = 67mm^2, and for the version that uses the 64KB 1T dRAM will be approximately 9.0mm$\times$10mm = 90 mm^2. The average power consumption for either design will be about 0.5W.

Because the *memory* uses the largest area and is the most difficult part of the design, two alternative memory designs were developed. The 16KB 4T dRAM is a conservative 4-transistor dynamic RAM designed as a low-risk option in case a higher density dRAM proved to be infeasible. This 4T dRAM is based on a cross-coupled n-channel cell. Data bits are in double-rail form, and reading is accomplished by precharging both data lines and then applying the word select. Writing is accomplished by driving the data lines to complementary values and applying the word select. The RAM performs a memory cycle on every clock cycle. In 1.2μm CMOS, it has an access time less than 20ns, and a cycle time of 25ns. The 64KB 1T dRAM is an aggressive, one-transistor-per-bit design that was completed in January 1990, and will be submitted for first full-scale fabrication on the MOSIS 1.2μm SCMOS run that is closing on 20 March 1990. (Several test structures have been fabricated and tested to verify the operation of circuits used in this dRAM.) The design of the dRAM is described in detail in section 4.7.

The *bootstrap ROM* is single-transistor mask programmable, and its read-cycle timing and organization is identical to that of the dRAM. The size listed, corresponding to 4K words, is much larger than necessary. The self-test, initialization, and bootstrap functions require approximately 600 words. However, because ROM

is denser than RAM, it may be useful in future systems to put standard subroutines (such as for floating-point arithmetic) in the ROM so as to save space in the RAM.

The 16-bit, microcode-driven *processor* is the only source of addresses in the node, and performs a memory cycle on every clock cycle. The processor datapath includes 24 general registers and 12 addressing and special registers. The instruction set is similar to that of other RISC processors, with 8 addressing modes for the move instructions, ALU operations including integer multiply, conditional branch instructions, a subroutine call, and control instructions. Projected performance using our present compilers and clock-by-clock microprogram simulation is 14 MIPS (16-bit operands).

The unusual features of the Mosaic processor are motivated by its use in a multicomputer node. The refresh and packet-interface address control are actually part of the processor, and the processor microcode interleaves instruction execution from four sources: two program contexts, refresh operations, and transfer between memory and the packet interface. The processor's address registers include two program counters, one for user code and the other for message-system control, with zero-time context switching between them. The two pointers and two limit registers for the send and receive queues are also in the address register set, together with the refresh address register. The remaining special registers control the interrupt status of the packet interface and the dx, dy, dz values in the header of messages that are being sent.

Either of two *routers* can be used. The 3D synchronous router consists of three cascaded 1D routing automata with a 4-bit-data path. A unidirectional external channel is 6 wires, consisting of 4 data lines, one escape bit for control codes, and the reverse flow-control signal. Bidirectional channels in each of 6 directions for 3D routing thus require a total of 72 external pins. The bandwidth per channel is one 4-bit data item each clock period, or 20MB/s. The 2D asynchronous router consists of two cascaded 1D routing automata with an 8-bit-data path. It is a variant on the FMRC2 routers developed for medium-grain multicomputers. A unidirectional external channel consists of 8 data lines, tail bit, request, and acknowledge. Bidirectional channels in each of 4 directions for 2D routing require 88 external pins. The bandwidth per channel in the 1.2μm CMOS technology will be approximately 80MB/s.

The *packet interface* includes 4 words of FIFO in each direction, the 16-bit-to-4/8-bit and 4/8-bit-to-16-bit conversion logic, and the logic that generates the message header on sending. The arbiter for deciding whether the system should perform memory refresh, channel data accesses, or processor access is also in the packet interface; the decisions that it generates are inputs to the processor microcode. The refresh signal is an input to the chip, and is bused through an entire array of Mosaic elements. The reason for synchronizing the refresh operation is that packets that are bound for a node that is refreshing would otherwise be blocked into the message

network, and block other messages that are in transit. Thus, one might as well refresh all of the nodes at once.

The Mosaic parts are quite modular, and can be assembled in a variety of floorplans. The principal internal interface is the memory bus, which consists of 16 data lines, 16 address lines, the write signal, and the clock and reset. In addition, there are several signals between the processor and packet interface, and two channels between the packet interface and the router.

2.1.3 Choice of network dimension

A Mosaic with 16,384 = 2^{14} nodes can be implemented either as a 128×128 two-dimensional routing mesh or a 32×32×16 three-dimensional routing mesh. The minimum bisection bandwidth of these two networks is the same: 128×80MB/s = 16×32×20MB/s = 10.24GB/s (in each direction). The significance of this figure of merit is that if message destinations are selected at random (a worst case), then half of the messages must traverse the bisection. Unless a substantial amount of internal buffering is available, the network becomes saturated at approximately half the bisection capacity.

The usual argument that the bisection limits the total volume of messages that can be produced and consumed by the nodes applies only to the case of randomly selected destinations. For a 16K-node network, either 2D or 3D, this limit is 1.25MB/s per node, or, for a typical message length of 20 Bytes, an average of one message each 16μs. In fact, simulations of the Mosaic runtime system's process-placement strategies show that the localization achieved in process placement reduces the number of messages that cross the bisection to substantially less than this worst case. It may well be possible for nodes to produce and consume 20B messages at rates in excess of one message each 4μs.

Analyses that assume the worst case of randomly selected message destinations favor a higher dimension network than is necessary for more localized message traffic. Our original plan for the Mosaic was to use a 32×32×16 three-dimensional routing mesh; however, it now appears that we will be able to save time and reduce risk by using a 2D network.

The latency using cut-through (wormhole) routing for a packet that is not blocked in the network is $T_{CT} = T_p D + L/B$, where T_p is the path-formation time through one router, D is the distance, L is the message length (*eg,* in Bytes), and B is the channel bandwidth (*eg,* in MB/s). For a 20Byte packet, the L/B term is 1μs for the 3D synchronous router and 0.25μs for the 2D asynchronous router. T_p is two clock periods, or 0.05μs for the 3D synchronous router; the longest path through this network is $D_{max} = 31 + 31 + 15$, so the maximum path-formation time is 3.85μs. T_p is expected to be 0.022μs for the 2D asynchronous router and the maximum path is $D_{max} = 127+127$, so the maximum path-formation time is 5.6μs. In fact, for localized

messages or longer messages (such as are encountered in program loading), the 2D network outperforms the 3D network.

Given the similar performance of these two networks, there are several other arguments in favor of using the 2D network:

1. The asynchronous 2D network eliminates the problems of coherent clock distribution required by the synchronous 3D network.
2. The protocol for the asynchronous 2D network is identical to that used in the Symult S2010 medium-grain multicomputer and the Intel Touchstone Delta prototype; thus, we would be able to employ the same host interfaces and other special devices (*eg,* displays) on either type of system.
3. The 2D packaging is considerably simpler, cheaper, and lower risk than the 3D packaging, and reduces the number of interboard connections by nearly a factor of four.

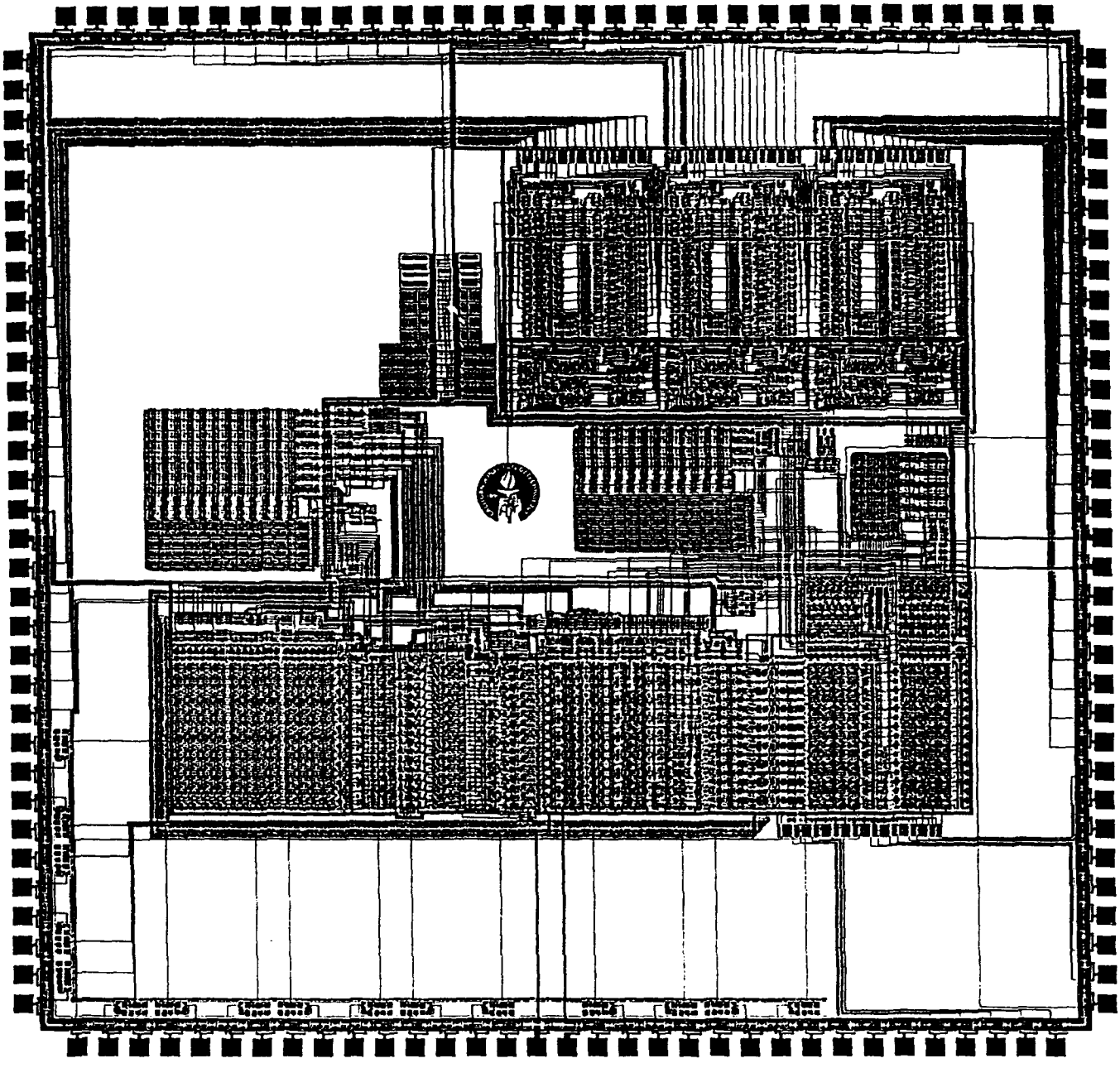
There is also an interesting issue of network scaling as it relates to our research agenda. The bisection argument presented above shows that the scaling of a mesh or torus network of given dimension is forced to the next higher dimension only when the radix (number of nodes on one dimension) becomes too large. The actual numbers show that 128 is close to the practical limit for the radix. Thus, if we can demonstrate that a 128×128 network and the localization accomplished by our runtime system still allow efficient execution with fully automatic process placement, we have also demonstrated that efficient execution would scale readily (with the problem size also scaling) to an $N = 128\times128\times128 = 2^{21}$-node system!

Another part of our long-term research agenda is to consider whether the third dimension should be reserved not for another dimension of mesh, but for long-distance connections; for example, a free-space optical shuffle. This consideration adds additional hesitancy to using the third dimension prematurely.

2.1.4 The Memoryless Mosaic chip

The *Memoryless Mosaic* chip has been a key part of our system-development strategy for the Mosaic. This chip (see the plot on the following page) is a complete Mosaic element except for the ROM and dRAM. It includes the Mosaic processor, packet interface, router, clock driver, and bus arbitration logic. The address and data buses are brought off of the chip; thus, the Memoryless Mosaic chip has allowed us to test the logic sections of the Mosaic under conditions in which the memory address and data are observable, and the memory data are controllable. It would otherwise be extremely difficult to diagnose internal problems in the Mosaic node, because the router, packet interface, and processor must function correctly in order to test them!

Extensive testing uncovered a design error in November 1989 in the first silicon of the Memoryless Mosaic, which was fabricated by MOSIS in 1.6μm SCMOS. The bug was in the packet-interface section, and was eventually traced to a missing 4λ×4λ



Memoryless Mosaic chip

patch of first-metal on one of the clock lines. This bug was not discovered during switch-level (Cosmos) simulation because the clock was supplied through an alternate path via a poly wire. This kind of error would ordinarily be expected merely to limit the speed of correct operation. However, in the Mosaic chip, it caused the control signals derived from the supposedly non-overlapping clock phases to overlap. The clock phases are generated on-chip, without the possibility of adjusting the non-overlapping time. As a result, several shift registers in the packet interface failed to operate correctly at any frequency. The detailed study of the FIFO section of the packet interface revealed ways of making it more robust, so this section was redesigned.

The corrected chip was submitted to MOSIS for 1.6μm SCMOS fabrication on 8 January 1990, and the revised parts were received on 14 March 1990. Preliminary tests indicate that the problem with the packet interface has been corrected, and the chips are fully functional.

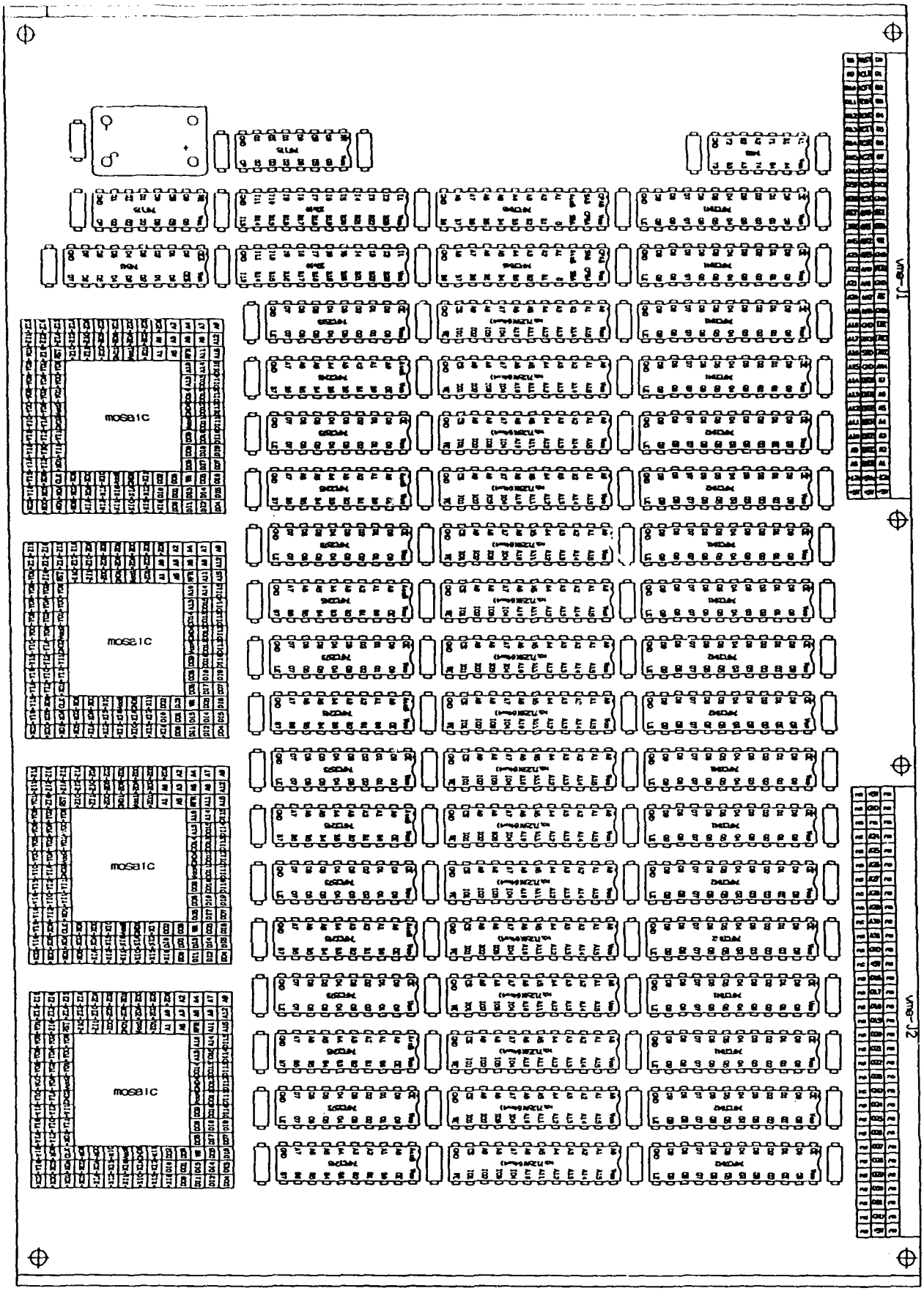
To test the logic sections of the Mosaic in the target 1.2μm SCMOS technology, a Memoryless Mosaic with a new pad frame was submitted to the MOSIS 1.2μm SCMOS run that closes on 20 March 1990.

2.1.5 Program-development systems

The other important application of the Memoryless Mosaic chip is to accelerate porting the programming systems, particularly the operating and runtime systems, from simulators to hardware. This bootstraping step is on the critical path of developing a useful system, and is also typically more difficult for multicomputers and other distributed-memory systems than it is for shared-memory systems. The observability and diagnosis of operating-system faults is problematic until the operating system is itself reliable.

We are able to get a head start on porting programming systems and application programs to the hardware, and also to simplify the operating-system-porting task, by building program-development systems that are based on the Memoryless Mosaic chips. These 6U VME boards (see the illustration on the following page) include 4 Memoryless Mosaics, which are connected by their channels in a 2×2 mesh. The external memory of each of the Memoryless Mosaics is 128KB of SRAM, which is two-ported to be read and written either by the Mosaic or through the VME interface. The clock rate is 20MHz. The SRAM is accessed by the Mosaic most of the time, and by the VME interface by cycle stealing. When the VME interface requests a memory access, the clock generator PAL stops the Mosaic clock signal for one clock period. While the Mosaic clock is stopped, the VME memory access is granted. The Mosaic clock and reset can also be controlled by memory-mapped storage locations. The logic design of these VME boards was just completed, and they are being sent to a commercial PCB house for layout and fabrication.

The completed boards will be plugged into the VME interfaces of our Sun



Mosaic program-development board

workstations or Symult S2010 systems. The host system will not only be able to load the memory of the nodes directly, but can also monitor program execution by examining the memory contents.

We expect in approximately three months to build another version of this program-development system for Memoryless Mosaics that use the asynchronous router. It will be possible to connect these boards together to form larger meshes, and to use these boards as host interfaces for larger Mosaic systems.

2.1.6 Packaging

Preliminary packaging designs for both 2D and 3D Mosaic systems have been completed. Both approaches use compression connectors to connect small circuit-board modules that are the testable and interchangeable units of manufacture, repair, and replacement. The Mosaic elements will be packaged and connected to the small circuit boards using TAB packaging.

The 4.2in×2.6in module for the 3D Mosaic contains 8 nodes in a 2×2×2 configuration with 320 external connections on two opposite edges. These modules are stacked between motherboards to create the 3D-packaging configuration. The 3D system is cooled by forced air in a direction parallel to the second dimension of routing.

The 4.2in×4.2in module for the 2D Mosaic contains 16 nodes in a 4×4 configuration with 400 external connections on all four edges. These modules are mounted to a power-distribution frame, and adjacent edges are joined by a single bridging connector.

2.1.7 Programming systems

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small.

The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system. Although Cantor was developed specifically for programming the Mosaic, Cantor programs can also be run today on medium-grain multicomputers, multiprocessors, sequential computers, and the Mosaic simulators.

The Mosaic can also be programmed at a lower level by using scaled-down versions of the C-based programming systems (Cosmic C, Reactive C) that we have developed for and used with medium-grain multicomputers.

These programming systems are quite stable and powerful. The continued improvement of these systems depends principally on progress in our related research efforts (see sections 3.1–3.4).

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

Our principal current research efforts with medium-grain multicomputers are aimed at new versions of our reactive-process programming systems and at advances in the performance of our mesh-routing chips. Our Caltech project continues to work closely with the DARPA-supported Touchstone project at Intel Scientific Computers. Our contributions include the architectural design, message-routing methods and chips, and system software. (See section 3.3 for a summary our current efforts with the Cosmic Environment and Reactive Kernel systems, and section 4.5 for a summary of our efforts with mesh-routing chips.)

The project operates several multicomputers: 8-node and 64-node Cosmic Cubes, a 128-node Intel iPSC/1, a 16-node Intel iPSC/2, and 32-node and 192-node Symult S2010 systems. The 192-node S2010 system is now the preferred machine for users. It is accessed through the Caltech Concurrent Supercomputer Facilities, and utilization has been at a level of approximately 90% of the available node-hours. All of these systems run very dependably.

Copies of the Cosmic Environment system have been distributed on request to approximately ten additional sites during this period, bringing the total copies distributed directly from the project to over 200.

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

3. Concurrent Computation

3.1 Runtime Systems for Fine-Grain Multicomputers

Nanette J. Boden, Chuck Seitz

We have been investigating several research problems that have emerged from our efforts to develop runtime systems for fine-grain multicomputers such as the Mosaic. These efforts are aimed at removing a number of restrictions on programming fine-grain multicomputers.

One easily understood example is the management of the node receive queue. A computation executing on the Mosaic will always consume a certain amount of space in each node for the runtime system itself, process code, process tables, and the persistent variables of the processes. The remaining space, which might be only one thousand bytes or so, can be used by the send and receive queues. Suppose that the computation involved a temporary "hot spot" that causes the receive queue in a node to overflow. When processes are able to exercise discretion in receiving messages selectively by their type or contents, they may not be able to consume the contents of the receive queue. In the present runtime systems, this is a deadlock, and the computation terminates.

It is, however, a serious flaw if a system with 1GB of memory, perhaps hundreds of MBs unused, might not be able to proceed because of a *local* fluctuation of a few hundred bytes. This problem also exists in medium-grain multicomputers, but is generally masked by the large size of the node memory. The solution is to export a part of the receive queue temporarily to another node, and, if necessary, to secondary storage. Indeed, several possible advances in system robustness and performance depend on introducing distributed solutions to resource-allocation problems.

Adding this kind of robustness to multicomputer programming systems is an example of the 80/20 rule: 80% of the sophistication in a runtime system is required to deal with the 20% residue of "difficult" cases and programs. Indeed, the compilation and runtime algorithms and heuristics for managing space without undue restrictions on the programmer, automatic process placement, managing the process-name space, determining code placement, and performing automatic code partitioning are remarkably subtle. They are also quite challenging when they must be implemented under serious constraints on both execution time and storage space.

Fast, efficient process placement is the key to several of these problems. Through analytical methods and simulation, we are exploring the spectrum from randomized to systematic node selection, that is, from methods depending entirely on randomization to methods that bias a random choice toward a local region or direction of growth, to methods that perturb a deterministic choice with "flip bits," to purely deterministic methods. A computation can be modeled for these purposes as an evolving population of processes. Each process on each timestep has a certain probability of creating another process or of self-destructing. Simulation approaches permit a realistic

complexity in the algorithms and heuristics being evaluated, and the incorporation of realistic machine models. However, these investigations are still somewhat removed from reality. Different resource allocation strategies may be more nearly optimal depending on the actual characteristics of application programs. In the analytical approach, the probabilities of process creation and of process self-destruction must be estimated; in simulation, randomized instances of "typical" programs must be used as input. The Mosaic system will allow us to refine the more promising approaches on full-scale application programs.

3.2 Composition Properties of Reactive-Process Programs

Nanette J. Boden, Chuck Seitz

The properties of adaptive-routing message systems, which may appear in future multicomputers, have numerous implications at levels ranging from the programming model to the the runtime support. The most attractive distributed approach to retaining message-order preservation is based on a reply-message protocol. It happens that this approach introduces a slightly stronger synchronization than the semantics supported in our current message-passing programming systems, in which message order is preserved only between pairs of communicating processes. The reply-message protocol allows the sending process to determine when a message is actually in the receive queue of the destination process, so that subsequent messages to "third parties" cannot lead to messages that precede the first message in the receive queue.

This stronger form of synchronization also has composition properties that are more uniform than those exhibited by our present message semantics. Curiously, it is also possible to obtain uniform composition properties by weakening our present message semantics into the unordered-message form of Actor semantics, but we can show that at least a weak form of message-order preservation is required to express certain computations efficiently. Uniform composition properties are not only desirable when attempting to reason about a program, they are also critical for being able to re-express a large process as a collection of small processes, either by hand or automatically. We are continuing to study the possibility of supporting this stronger (but compatible) form of message-order preservation in future systems.

3.3 The Cosmic Environment and Reactive Kernel

Wen-king Su, Jakov Seizovic, Chuck Seitz, Joe Beckenbach, Christopher Lee

Our plans for the development of new versions of the Cosmic Environment host runtime system and the Reactive Kernel node operating system were outlined in our previous semiannual technical report, and the work is in progress.

Version 7.2 of the Cosmic Environment has matured after enduring more than two years of academic and commercial applications. Based on our experiences with the Cosmic Environment, we are now in the position to suggest and implement major changes in the internal structure of the Cosmic Environment. One of the problems in

version 7.2 is the centralized multicomputer allocation and bookkeeping mechanism that places the Cosmic Environment at the mercy of network conditions. We have designed a robust distributed mechanism in which allocation is performed in the host of the multicomputer itself. Thus, the multicomputer would be inaccessible only when its host is inaccessible. We have also demonstrated a technique that increases the Cosmic Environment communication bandwidth from 40Kbytes/second to 300Kbytes/second with a small increase in message latency. We eliminate the need to perform extra handshakes across slow ethernet links by shifting the burden of buffering messages from the multicomputer's host machine to the user's host machine. We have also found a way to increase the message delivery rate for selected user processes, such as a frame buffer controller, by allowing the process to be merged with the message switcher process, thus saving one communication cycle and context-switch time for each message.

3.4 The Page Kernel

Craig S. Steele, Chuck Seitz

The previously-described "Page Kernel" (PK) concurrent programming environment is an evolutionary variant of the reactive kernel (RK). PK utilizes the virtual-memory capabilities of second-generation medium-grain multicomputers to render message origination and receipt implicit, and to move the low-level management of data sharing from the programmer to the kernel. Continuing development of the PK has resulted in simplification of the programming model and extension of its capabilities.

The executable unit is the *action*, a light-weight reactive process scheduled in response to modification of associated data structures (*blocks*). The programmer is responsible for writing code to specify which data blocks are accessible to each of the actions. Defining the multiple address spaces of the actions and coding the operations of the actions is the programmer's task; action scheduling and data communication are handled by the kernel.

Another common function appropriated to the kernel is the management of mutually-exclusive writing to data blocks shared by multiple actions. Rather than locking data with potential write conflicts, actions are allowed to proceed to completion before actual conflicts are evaluated. If an action is excluded from writing its results to a shared data block due to another action's access, it fails and none of its results are written to any data block. The action is undone with no visible effect, and it is rescheduled for later execution. This mechanism involves considerable data copying and duplication, but the additional cost is quite modest with second-generation multicomputer communications hardware; for example, it incurs about 25% in increased execution time on the Symult S2010. This implementation allows greater concurrency for problems with more potential than actual conflicts.

The PK is expected to be an attractive alternative programming environment for problems such as iterative optimization, in which the mechanics of distributing and

updating shared data structures may obscure the relative simplicity of a concurrent algorithm.

3.5 A C-Based Concurrent Programming Language For Multicomputers

Marcel van der Goot, Alain Martin

As described in the previous semi-annual report, we are defining and implementing a concurrent programming language for message-passing multicomputers. We have chosen C as the basis for the sequential parts of the language; the extensions that support concurrent programming include processes and CSP-like communication primitives. A first implementation, consisting of a compiler and a small runtime system, was finished in February 1990. The compiler takes our language as input and has standard (ANSI) C as target; the runtime system contains functions to support the concurrent execution of processes. The output of our compiler is compiled for a SUN workstation where it is executed as a single UNIX process.

So far, the compiler has been used by the students in a concurrent programming class, and to write a (functional) simulation of the asynchronous microprocessor. Since the specification of the microprocessor is in a language similar to ours, the simulation program was relatively easy to write. Currently, we are working on documentation and on porting the implementation to an actual multicomputer (the Symult S2010, or any other multicomputer that runs CE/RK), together with some reorganization of the compiler. We expect that neither the compiler nor the runtime system will require much rewriting for this parallel implementation.

4. VLSI Design

4.1 Automatic Synthesis of Asynchronous Circuits

Dražen Borković, Steve Burns, Alain J. Martin

The second generation of synthesis tools that we envision will integrate simulation, performance evaluation, and optimization (transistor sizing). The designer will be able (or perhaps will be required) to make choices at different stages of the synthesis based on the results of the previous stage. As a first step toward such a system, we are designing a program for the synthesis of straightline program into CMOS chips. The final program will include automatic cell synthesis, transistor sizing, placement and routing.

4.2 Cache Memory for an Asynchronous Microprocessor

Alain J. Martin, José A. Tierno

The design of a direct-mapped instruction cache for an asynchronous microprocessor is almost completed. The circuit has been derived from a high-level specification, and both control circuitry and RAM array are completely delay-insensitive with the exception of isochronic forks. Special attention was paid to the design of the RAM cell, to optimizing the signaling protocol, and to eliminating unnecessary transitions and completion trees. The full (conservative) implementation requires 13 transistors per memory cell, of which 3 can be eliminated at the expense of a bigger delay. The RAM array has a special read-write cycle. The rest of the control was designed around this cell, since the bottleneck in throughput will be in the access to the RAM array.

4.3 Testing Self-Timed Circuits

Pieter Hazewindus, Alain J. Martin

We are studying the problem of increasing the fault coverage of our designs by adding testing circuitry to the circuits. The fault model we use is the single stuck-at fault model. For any non-redundant circuit, if we can set and observe the value of each state-holding element, then all faults are testable. Since it is infeasible to connect every state-holding element to a pad, we use as testing circuitry a simple queue that connects all state-holding elements. For such a scheme, the only untestable faults would be located in the queue.

We have designed a testing queue that has twelve transistors per stage. For normal circuit operation, the penalty for having the testing circuitry is just one pass gate, so that the decrease in performance is minor. For the control of the microprocessor, the number of transistors in the clocked testing queue is about half the total number of transistors. We are trying to reduce the size of the testing queue by reducing the number of state-holding elements observed. It seems that possible global optimizations, at the program level or otherwise, are rare, but some ad hoc or local optimizations are possible.

4.4 Sizing the Transistors of Asynchronous Circuits

Steve Burns, Alain Martin

We have developed a method of optimally sizing the transistors contained in the asynchronous circuits that we construct by systematic transformation from concurrent programs. These transistors are sized optimally if the sizes minimize the time needed to operate the circuit, minimize the energy required to operate the circuit, or minimize some other metric of performance.

The concerns of performance optimization in asynchronous circuits are quite different than those of synchronous (clocked) circuits. In the synchronous cases, the main task is to determine and then speed up the slowest or critical path through the combinational logic that connect the clocked latches. This is in order to maintain correctness, since for correct operation, the combinational logic must complete before the clock changes.

In the asynchronous circuits derived using our synthesis method, the circuits work correctly regardless of delays in the primitive gates. For most applications (i.e., those without hard real-time deadlines), it is not necessary to optimize the worst case (or even to know what it is). Rather, it is the average case that determines a circuit's performance. While an operation that requires twice the time but occurs only once every one hundred operations is catastrophic to a synchronous design, it only decreases the performance of our asynchronous circuits by one percent.

Much of the computation involved in the performance analysis of synchronous circuits, in particular that of determining the critical paths induced by unusual data patterns, can be avoided by using our asynchronous methodology. An average or typical operation sequence is specified and a performance metric is determined based on that sequence. Since our asynchronous circuits work correctly regardless of gate delays, it turns out that the performance metric is a convex function of the transistor sizes and thus each local minimum to the function is also a global minimum. The techniques of convex non-linear programming can be used to find these optimal sizes. A C program has been written to perform these calculations. Optimal transistor sizes for a typical 40 transistor circuit can be obtained in under 10 seconds on a SUN 3/60.

4.5 Fast Self-Timed Mesh-Routing Chips

Chuck Seitz

A new version in the FMRC series of mesh-routing chips has been laid out, verified by switch-level simulation, and sent to fabrication for the 1.2μm MOSIS SCMOS run that is scheduled to close on 20 March 1990. Previous FMRC chips have been fabricated in 1.6μm SCMOS, and operate at 65MB/s, but exhibit some reliability problems when the aggregate throughput of the chip's 5 output channels exceeds about 250MB/s. This reliability problem was traced by analysis and simulation to collapse of the internal power supply under these demanding conditions; thus, it is properly a failure of the packaging rather than of the chip design.

This 132-pin chip devotes the 20 lowest-inductance PGA-package pins to Vdd and GND. It was not deemed to be practical for the immediate application (the Intel Touchstone Delta prototype) to increase the pinout to allow additional Vdd and GND pins; however, it was considered to be desirable to increase the speed to in excess of 80MB/s. Intel is tooling a special package whose internal power and ground planes reduce the inductance of the power distribution from the package by a factor of approximately two. However, in designing new pad circuits and pad frame for the FMRC, I decided to take all available measures that might improve the reliability of these chips.

With the support and encouragement of Wes Hansford at MOSIS, we were able to reduce the pad pitch from 6 mils to 5 mils, with a 90μm square pad. The resistance of the pad-power ring was reduced in comparison with our standard 1.6μm pads by a factor of nearly four by a combination of increased width and use of both metal layers where possible. The peak pad-drive current was reduced to about 0.75 of its value for the 1.6μm pad drivers, and the p/n ratio was reduced from 5/3 (which produces symmetrical transitions in the 1.6μm process) to 4/3 to compensate for the transistors being farther into velocity saturation. Additional speed in the core of the router will more than make up for the slightly slower pads. These measures reduce the total current and ohmic drops; they also decrease di/dt effects of the package-pin inductance. As additional measures to reduce the di/dt effects, nearly all of the "white space" in this pad-limited design was used to add power-decoupling capacitance, which is believed to be more than 500pF. The drive of the output pads was also tuned to minimize di/dt. (A plot of the chip is shown on the following page.)

The design and layout of a successor to the FMRC is underway.

4.6 Adaptive Routing in Multicomputer Networks

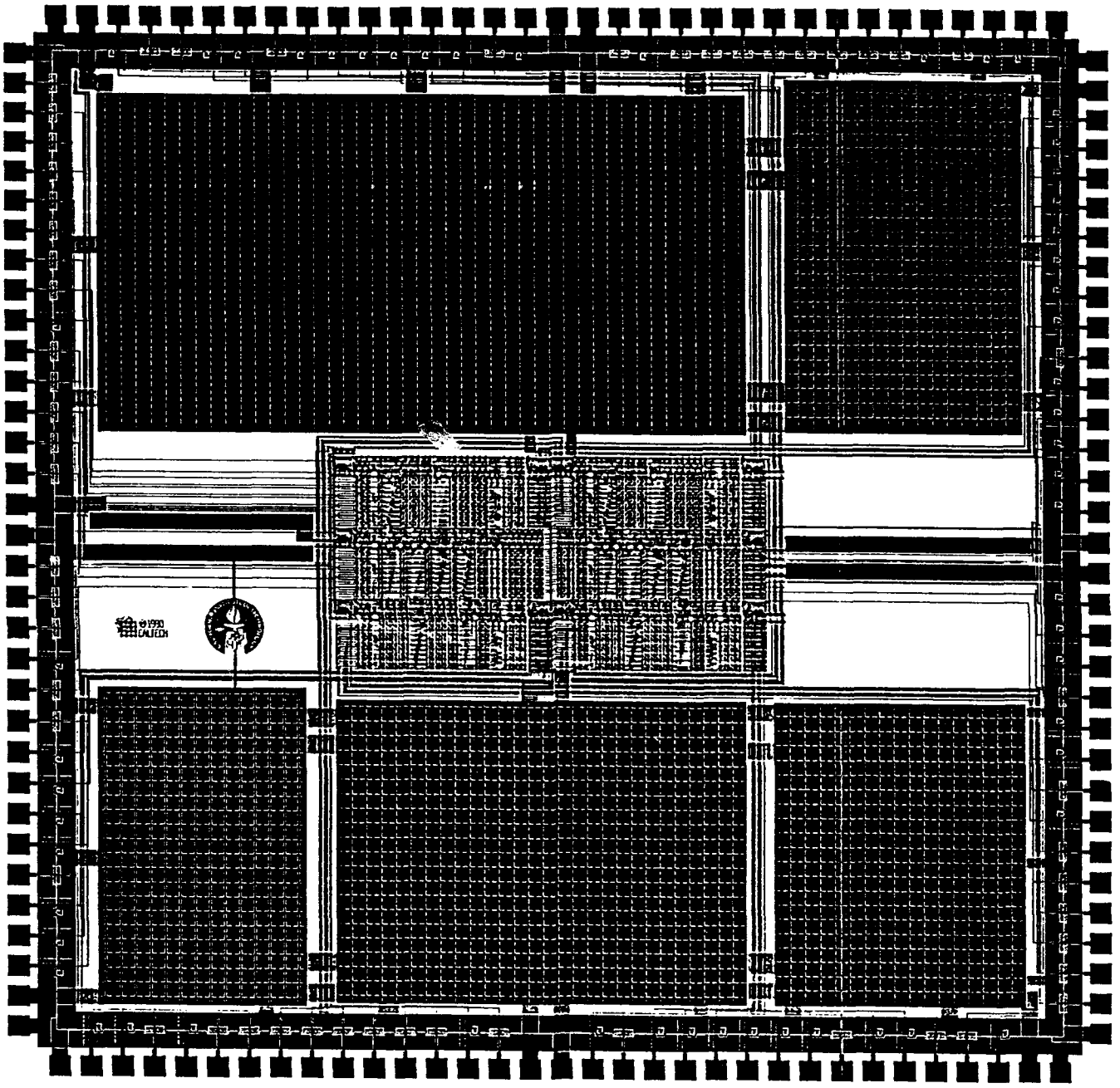
Mike Pertel, Chuck Seitz

Previous theoretical studies of adaptive multipath routing are being continued, and an adaptive router for the Mosaic is being designed. Under simulation, adaptive routers have exhibited superior throughput, traffic diffusion, and fault tolerance, as compared with oblivious routers. Further simulation is being used to refine and simplify the routing discipline before committing to silicon.

4.7 High-Density Mosaic dRAM

Don Speck

Multicomputers have been tending toward more memory per node as they get faster, and Mosaic is no exception. Having more never hurts, and it extends the application range and ease of programming. Therefore, when the Mosaic C design began, design of a dense dynamic memory began with it. The simulation and layout of a 32K×16 dynamic RAM is now complete, and ready for first fabrication in the $\lambda = 0.6\mu$m MOSIS SCMOS process. This 64KB memory is half as much as in a Cosmic Cube



FMRC2.2 mesh-routing chip

node, and is the largest power-of-2 size smaller than Mosaic C's addressing limit. It is also the largest area ($13470\lambda \times 11974\lambda$) that doesn't need repeaters in all of the wires, and is about 75% of the total chip area (which is how much was budgeted for RAM).

The design of this dynamic RAM attempted to simultaneously optimize area, energy, speed, and noise immunity. Small area is the primary reason for choosing a one-transistor-per-bit style instead of something easier to analyze (otherwise why bother?), and it also helps shorten the long wires that contribute to delay and power consumption. Power dissipation is at a premium in large ensembles of closely-packed nodes, and the only way to significantly reduce total chip power is to reduce the power supply voltage to 4V or even 3.3V; for wafer-scale packaging, 2.5V would be required. In addition, a safety factor of plus or minus 20% is needed to allow for process variations. Over such a wide operating range, it is not possible to meet a fixed speed and noise immunity specification regardless of voltage, nor is it necessary. The RAM only has to keep up with the processor, whose speed varies with voltage, and the noise immunity has to exceed noise generation, which also varies with voltage (quadratically in the case of resistive drops, less than linearly for the backgate component of threshold variation).

To accommodate the processor on the same chip and have access to the smallest feature size of the day, the RAM uses a standard MOSIS logic process and is designed to satisfy all of the Magic DRC rules for the most restrictive process, in either nwell or pwell. (The latter disallows boosted signals). The best bit storage capacitor in that process is an enhancement-mode MOS capacitor, which has low charge-storage density and cannot store the full power supply voltage range. These are the same limitations that the early commercial dRAM designers faced, so the support circuits that worked well then also turn out to be good choices for this RAM.

Making the cell capacitor large to compensate for low charge-storage density is subject to diminishing returns. The bitline length and capacitance grow with the cell capacitor. Larger depletion regions collect more minority carriers from alpha-particle strikes. Larger MOS capacitors are slower and cannot be charged as fully in the time available; even with a modest capacitor size, writing has to start very early to approach full charge. Beyond some point, the area is better used elsewhere, such as for more sense amps, and this point is about $64\lambda^2$. This is just big enough for a half-sized dummy cell to be feasible. A full-sized dummy cell would need a half-charge reference voltage, which is not $V_{dd}/2$ due to the MOS capacitor threshold. At the lowest operating voltage, the capacitor cannot even store $V_{dd}/2$.

The small bitcell has room for only one bitline through it, and, without a second poly layer, this mandates an open bitline arrangement. Open bitlines require more careful matching of noises on opposite sides of the sense amp than do folded bitlines. There is no place to put transistors to short together bitline pairs; instead, oversized prechargers short all bitlines to an equilibration line, which then connects to Vdd only at its center tap, to equalize power glitches. The substrate has similar equilibration

wires center-tapped to ground spaced 16 bitlines apart, taking up about 5% of the RAM area.

These noises can't be perfectly matched, so it is advisable to make the readout voltage large in comparison, in this case by keeping the bitlines short — only 32 bitcells — resulting in a 6:1 bitline-to-cell capacitance ratio. The sense amplifiers have to be small and simple to avoid dominating the total area, but a simple cross-coupled pair suffices when the signal voltage is large and bitline capacitance is low. Low bitline capacitance also makes full-V_{dd} precharge affordable, which is needed anyway because at the lower supply voltages (*eg,* 2V), $V_{dd}/2$ precharge wouldn't be enough to turn on the sense-amp transistors. The column-select transistors double as cascodes that isolate the bitlines from the I/O line capacitance until the bitlines fall a threshold below V_{dd}. Area-consuming level-restore circuits are not needed on the sense amps, because the storage capacitor cannot store full voltage levels, but one is used on the I/O lines in case the bitlines fall far enough for the cascodes to slowly leak.

There are 8192 sense amps but only 16 bits need be read or written at once. There is neither need nor room for a read/write amplifier per sense amp. Fortunately, the bitline pitch is larger than minimum metal spacing, leaving enough room to intersperse column select lines from a shared column decoder, controlling the multiplexing of 64 sense amplifiers onto 2 read/write amplifiers via I/O lines perpendicular to the bitlines. Space has to be made periodically for read/write amplifiers to keep the I/O line capacitance low enough to be driven quickly by the sense amplifiers, providing a good place to insert row decoders that keep the wordlines short enough to run in poly without metal strapping. Strapping the wordlines would have increased bitline capacitance by 10%; the increase in bitcell area needed to counteract this would have been more than the row decoder area.

The short bitlines and wordlines divide the RAM into 8 by 8 banks. To keep each data bus wire under 12000λ, only 2 bits connect to each bank, so 8 banks must power up on each cycle. About half of the power consumed goes into address distribution, decoding, and clocks. If prechargers in unselected banks were turned on and off every cycle, that would add 25% to the power consumption (all from the clocks); instead, the first three address bits control them. Precharge turn-on needs to wait anyway until the wordlines finish falling; hence, it is controlled by a delay line. This obviates any need for a second clock phase, saving clock wiring and its attendant power dissipation.

The sense amplifiers are on a 10.5λ pitch; this demands that they be connected common-source to a current generator. The amount of current a sense amp receives depends both on its own bitline voltages and on the bitline voltages of other sense amps. Initial current is set low, so that sense amps receiving the most current get no more than is safe, although this means that some sense amps receive none at first. As the sense amps with an early start develop signal, current is ramped up until all sense amps are conducting. Further current increases are delayed until the late starters

catch up, then a larger current ramps up. The sense timing generator ramps up voltages on transistor gates via current mirrors, and fits underneath the row decoder address wires along with a delay line to simulate the wordline delay.

AREA BREAKDOWN:

bitcells 61%

sense amps, prechargers, dummy cells 15%

power/ground wires 11%

row decoders 8%

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports

16 March 1990

Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check for a minimum of $50.00 drawn on a U.S. bank in U.S. currency, payable to CALTECH.

CS-TR-90-03	$3.00	Program Composition Project Chandy, K Mani with Stephen Taylor, Carl Kesselman and Ian Foster
CS-TR-90-02	$2.00	Limitations to Delay-Insensitivity in Asynchronous Circuits Martin, Alain J
CS-TR-90-01	$3.00	Properties of the V-C Dimension, MS Thesis Fyfe, Andrew
CS-TR-89-12	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-89-11	$9.00	Reactive-Process Programming and Distributed Discrete-Event Simulation, PhD Thesis Su, Wen-King
CS-TR-89-10	$7.00	Silicon Models of Early Audition, PhD Thesis Lazarro, John
CS-TR-89-09	$15.00	Framework for Adaptive Routing in Multicomputer Networks, PhD Thesis Ngai, John
CS-TR-89-07	$6.00	Constraint Methods for Neural Networks and Computer Graphics, PhD Thesis Platt, John
CS-TR-89-06	$1.00	First Asynchronous Microprocessor: The Test Results Martin, Alain J, Steven M Burns, T K Lee, Drazen Borkovic, and Pieter J Hazewindus
CS-TR-89-05	$2.00	Essence of Distributed Snapshots Chandy, K Mani
CS-TR-89-04	$5.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-89-03	$3.00	Feature-oriented Image Enhancement with Shock Filters, I Rudin, Leonid I with Stanley Osher
CS-TR-89-02	$3.00	Design of an Asynchronous Microprocessor Martin, Alain J
CS-TR-89-01	$4.00	Programming in VLSI From Communicating Processes to Delay-insensitive Circuits Martin, Alain J
CS-TR-88-22	$2.00	Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm Su, Wen-King and Charles L Seitz
CS-TR-88-21	$3.00	Winner-Take-All Networks of O(N) Complexity Lazzaro, John, with S Ryckebusch, M A Mahowald and C A Mead
CS-TR-88-20	$7.00	Neural Network Design and the Complexity of Learning Judd, J Stephen
CS-TR-88-19	$5.00	Controlling Rigid Bodies with Dynamic Constraints Barzel, Ronen
CS-TR-88-18	$3.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-88-17	$3.00	Constrained Differential Optimization for Neural Networks Platt, John C and Alan H Barr

CS-TR-88-16	$3.00	Programming Parallel Computers Chandy, K Mani
CS-TR-88-15	$13.00	Applications of Surface Networks to Sampling Problems in Computer Graphics, PhD Thesis Von Herzen, Brian
CS-TR-88-14	$2.00	Syntax-directed Translation of Concurrent Programs into Self-timed Circuits Burns, Steven M and Alain J Martin
CS-TR-88-13	$2.00	Message-Passing Model for Highly Concurrent Computation Martin, Alain J
CS-TR-88-12	$4.00	Comparison of Strict and Non-strict Semantics for Lists, MS Thesis Burch, Jerry R
CS-TR-88-11	$5.00	Study of Fine-Grain Programming Using Cantor, MS Thesis Boden, Nanette J
CS-TR-88-10	$3.00	Reactive Kernel, MS Thesis Seizovic, Jacov
CS-TR-88-07	$3.00	Hexagonal Resistive Network and the Circular Approximation Feinstein, David I
CS-TR-88-06	$3.00	Theorems on Computations of Distributed Systems Chandy, K Mani
CS-TR-88-05	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-88-04	$3.00	Cochlear Hydrodynamics Demystified Lyon, Richard F and Carver A Mead
CS-TR-88-03	$4.00	PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics, MS Thesis Gupta, Rajiv
CS-TR-88-02	$4.00	Automated Compilation of Concurrent Programs into Self-timed Circuits, MS Thesis Burns, Stephen M
CS-TR-88-01	$3.00	C Programmer's Abbreviated Guide to Multicomputer Programming Seitz, Charles, Jakov Seizovic and Wen-King Su
5258:TR:88	$3.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5256:TR:87	$2.00	Synthesis Method for Self-timed VLSI Circuits Martin, Alain. *(current supply only: see Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design 224-229, Oct'87)*
5253:TR:88	$2.00	Synthesis of Self-Timed Circuits by Program Transformation Burns, Steven M and Alain J Martin
5251:TR:87	$2.00	Conditional Knowledge as a Basis for Distributed Simulation Chandy, K Mani and Jay Misra
5250:TR:87	$10.00	Images, Numerical Analysis of Singularities and Shock Filters, PhD Thesis Rudin, Leonid Iakov
5249:TR:87	$6.00	Logic from Programming Language Semantics, PhD Thesis Choo, Young-il
5247:TR:87	$6.00	VLSI Concurrent Computation for Music Synthesis, PhD Thesis Wawrzynek, John
5246:TR:87	$3.00	Framework for Adaptive Routing Ngai, John Y and Charles L Seitz
5244:TR:87	$3.00	Multicomputers Athas, William C and Charles L Seitz
5243:TR:87	$5.00	Resource-Bounded Category and Measure in Exponential Complexity Classes, PhD Thesis Lutz, Jack H

5242:TR:87	$8.00	Fine Grain Concurrent Computations, PhD Thesis Athas, William C
5241:TR:87	$3.00	VLSI Mesh Routing Systems, MS Thesis Flaig, Charles M
5240:TR:87	$2.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5239:TR:87	$3.00	Trace Theory and Systolic Computations Rem, Martin
5238:TR:87	$7.00	Incorporating Time in the New World of Computing System, MS Thesis Poh, Hean Lee
5236:TR:86	$4.00	Approach to Concurrent Semantics Using Complete Traces, MS Thesis Van Horn, Kevin S
5235:TR:86	$4.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5234:TR:86	$3.00	High Performance Implementation of Prolog Newton, Michael O
5233:TR:86	$3.00	Some Results on Kolmogorov-Chaitin Complexity, MS Thesis Schweizer, David Lawrence
5232:TR:86	$4.00	Cantor User Report Athas, W C and C L Seitz
5230:TR:86	$24.00	Monte Carlo Methods for 2-D Compaction, PhD Thesis Mosteller, R C
5229:TR:86	$4.00	anaLOG - A Functional Simulator for VLSI Neural Systems, MS Thesis Lazzaro, John
5228:TR:86	$3.00	On Performance of k-ary n-cube Interconection Networks Dally, Wm J
5227:TR:86	$18.00	Parallel Execution Model for Logic Programming, PhD Thesis Li, Pey-yun Peggy
5223:TR:86	$15.00	Integrated Optical Motion Detection, PhD Thesis Tanner, John E
5221:TR:86	$3.00	Sync Model: A Parallel Execution Method for Logic Programming Li, Pey-yun Peggy and Alain J Martin. *(current supply only: see Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86)*
5220:TR:86	$4.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5215:TR:86	$2.00	How to Get a Large Natural Language System into a Personal Computer Thompson, Bozena H and Frederick B Thompson
5214:TR:86	$2.00	ASK is Transportable in Half a Dozen Ways Thompson, Bozena H and Frederick B Thompson
5212:TR:86	$2.00	On Seitz' Arbiter Martin, Alain J
5210:TR:86	$2.00	Compiling Communicating Processes into Delay-Insensitive VLSI Circuits Martin, Alain. *(current supply only: see Distributed Computing v 1 no 4 (1986))*
5207:TR:86	$2.00	Complete and Infinite Traces: A Descriptive Model of Computing Agents van Horn, Kevin
5205:TR:85	$2.00	Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity Schweizer, David and Yaser Abu-Mostafa
5204:TR:85	$3.00	An Inverse Limit Construction of a Domain of Infinite Lists Choo, Young-Il

5202:TR:85	$15.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5200:TR:85	$18.00	ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation, PhD Thesis Whelan, Dan
5198:TR:85	$8.00	Neural Networks, Pattern Recognition and Fingerprint Hallucination, PhD Thesis Mjolsness, Eric
5197:TR:85	$7.00	Sequential Threshold Circuits, MS thesis Platt, John
5195:TR:85	$3.00	New Generalization of Dekker's Algorithm for Mutual Exclusion Martin, Alain J. *(current supply only: see Information Processing Letters 23 295-297 1986)*
5194:TR:85	$5.00	Sneptree - A Versatile Interconnection Network Li, Pey-yun Peggy and Alain J Martin
5193:TR:85	$2.00	Delay-insensitive Fair Arbiter Martin, Alain J
5190:TR:85	$3.00	Concurrency Algebra and Petri Nets Choo, Young-il
5189:TR:85	$10.00	Hierarchical Composition of VLSI Circuits, PhD Thesis Whitney, Telle
5185:TR:85	$11.00	Combining Computation with Geometry, PhD Thesis Lien, Sheue-Ling
5184:TR:85	$7.00	Placement of Communicating Processes on Multiprocessor Networks, MS Thesis Steele, Craig
5179:TR:85	$3.00	Sampling Deformed, Intersecting Surfaces with Quadtrees, MS Thesis Von Herzen, Brian P
5178:TR:85	$9.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5174:TR:85	$7.00	Balanced Cube: A Concurrent Data Structure Dally, William J and Charles L Seitz
5172:TR:85	$6.00	Combined Logical and Functional Programming Language Newton, Michael
5168:TR:84	$3.00	Object Oriented Architecture Dally, Bill and Jim Kajiya
5165:TR:84	$4.00	Customizing One's Own Interface Using English as Primary Language Thompson, B H and Frederick B Thompson
5164:TR:84	$13.00	ASK French - A French Natural Language Syntax, MS Thesis Sanouillet, Remy
5160:TR:84	$7.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5153:TR:84	$6.00	VLSI Architecture for Sound Synthesis Wawrzynek, John and Carver Mead
5157:TR:84	$15.00	Bit-Serial Reed-Solomon Decoders in VLSI, PhD Thesis Whiting, Douglas
5147:TR:84	$4.00	Networks of Machines for Distributed Recursive Computations Martin, Alain and Jan van de Snepscheut
5143:TR:84	$5.00	General Interconnect Problem, MS Thesis Ngai, John
5140:TR:84	$5.00	Hierarchy of Graph Isomorphism Testing, MS Thesis Chen, Wen-Chi
5139:TR:84	$4.00	HEX: A Hierarchical Circuit Extractor, MS Thesis Oyang, Yen-Jen

5137:TR:84	$7.00	Dialogue Designing Dialogue System, PhD Thesis Ho, Tai-Ping
5136:TR:84	$5.00	Heterogeneous Data Base Access, PhD Thesis Papachristidis, Alex
5135:TR:84	$7.00	Toward Concurrent Arithmetic, MS Thesis Chiang, Chao-Lin
5134:TR:84	$2.00	Using Logic Programming for Compiling APL, MS Thesis Derby, Howard
5133:TR:84	$13.00	Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems, PhD Thesis Lin, Tzu-mu
5132:TR:84	$10.00	Switch Level Fault Simulation of MOS Digital Circuits, MS Thesis Schuster, Mike
5129:TR:84	$5.00	Design of the MOSAIC Processor, MS Thesis Lutz, Chris
5128:TM:84	$3.00	Linguistic Analysis of Natural Language Communication with Computers Thompson, Bozena H
5125:TR:84	$6.00	Supermesh, MS Thesis Su, Wen-King
5123:TR:84	$14.00	Mossim Simulation Engine Architecture and Design Dally, Bill
5122:TR:84	$8.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5114:TM:84	$3.00	ASK As Window to the World Thompson, Bozena, and Fred Thompson
5112:TR:83	$22.00	Parallel Machines for Computer Graphics, PhD Thesis Ulner, Michael
5106:TM:83	$1.00	Ray Tracing Parametric Patches Kajiya, James T
5104:TR:83	$9.00	Graph Model and the Embedding of MOS Circuits, MS Thesis Ng, Tak-Kwong
5094:TR:83	$2.00	Stochastic Estimation of Channel Routing Track Demand Ngai, John
5092:TM:83	$2.00	Residue Arithmetic and VLSI Chiang, Chao-Lin and Lennart Johnsson
5091:TR:83	$2.00	Race Detection in MOS Circuits by Ternary Simulation Bryant, Randal E
5090:TR:83	$9.00	Space-Time Algorithms: Semantics and Methodology, PhD Thesis Chen, Marina Chien-mei
5089:TR:83	$10.00	Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits Lin, Tzu-Mu and Carver A Mead
5086:TR:83	$4.00	VLSI Combinator Reduction Engine, MS Thesis Athas, William C Jr
5082:TR:83	$10.00	Hardware Support for Advanced Data Management Systems, PhD Thesis Neches, Philip
5081:TR:83	$4.00	RTsim - A Register Transfer Simulator, MS Thesis Lam, Jimmy
5074:TR:83	$10.00	Robust Sentence Analysis and Habitability Trawick, David

5073:TR:83	$12.00	Automated Performance Optimization of Custom Integrated Circuits, PhD Thesis Trimberger, Steve
5065:TR:82	$3.00	Switch Level Model and Simulator for MOS Digital Systems Bryant, Randal E
5054:TM:82	$3.00	Introducing ASK, A Simple Knowledgeable System Conf on App'l Natural Language Processing Thompson, Bozena H and Frederick B Thompson
5051:TM:82	$2.00	Knowledgeable Contexts for User Interaction Proc Nat'l Computer Conference Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho
5035:TR:82	$9.00	Type Inference in a Declarationless, Object-Oriented Language, MS Thesis Holstege, Eric
5034:TR:82	$12.00	Hybrid Processing, PhD Thesis Carroll, Chris
5033:TR:82	$4.00	MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual Schuster, Mike, Randal Bryant and Doug Whiting
5029:TM:82	$4.00	POOH User's Manual Whitney, Telle
5018:TM:82	$2.00	Filtering High Quality Text for Display on Raster Scan Devices Kajiya, Jim and Mike Ullner
5017:TM:82	$2.00	Ray Tracing Parametric Patches Kajiya, Jim
5015:TR:82	$15.00	VLSI Computational Structures Applied to Fingerprint Image Analysis Megdal, Barry
5014:TR:82	$15.00	Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture, PhD Thesis Lang, Charles R Jr
5012:TM:82	$2.00	Switch-Level Modeling of MOS Digital Circuits Bryant, Randal
5000:TR:82	$6.00	Self-Timed Chip Set for Multiprocessor Communication, MS Thesis Whiting, Douglas
4684:TR:82	$3.00	Characterization of Deadlock Free Resource Contentions Chen, Marina, Martin Rem, and Ronald Graham
4655:TR:81	$20.00	Proc Second Caltech Conf on VLSI Seitz, Charles, ed.
4090-TR-80	$3.00	VLSI Based Real-Time Hidden Surface Elimination Display System, MS Thesis Demetrescu, Stefan G
3760:TR:80	$10.00	Tree Machine: A Highly Concurrent Computing Environment, PhD Thesis Browning, Sally
3759:TR:80	$10.00	Homogeneous Machine, PhD Thesis Locanthi, Bart
3710:TR:80	$10.00	Understanding Hierarchical Design, PhD Thesis Rowson, James
3340:TR:79	$26.00	Proc. Caltech Conference on VLSI (1979) Seitz, Charles, ed
2276:TM:78	$12.00	Language Processor and a Sample Language Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

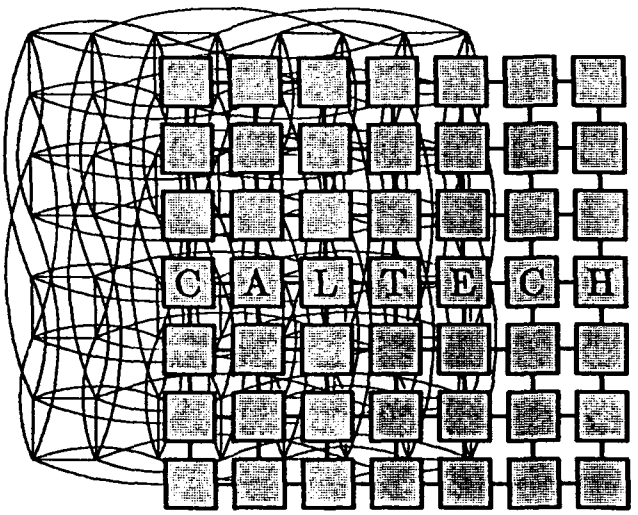
_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

___CS-TR-90-03	___CS-TR-88-12	___5238:TR:87	___5195:TR:85	___5134:TR:84	___5051:TM:82
___CS-TR-90-02	___CS-TR-88-11	___5236:TR:86	___5194:TR:85	___5133:TR:84	___5035:TR:82
___CS-TR-90-01	___CS-TR-88-10	___5235:TR:86	___5193:TR:85	___5132:TR:84	___5034:TR:82
___CS-TR-89-12	___CS-TR-88-07	___5234:TR:86	___5190:TR:85	___5129:TR:84	___5033:TR:82
___CS-TR-89-11	___CS-TR-88-06	___5233:TR:86	___5189:TR:85	___5128:TM:84	___5029:TM:82
___CS-TR-89-10	___CS-TR-88-05	___5232:TR:86	___5185:TR:85	___5125:TR:84	___5018:TM:82
___CS-TR-89-09	___CS-TR-88-04	___5230:TR:86	___5184:TR:85	___5123:TR:84	___5017:TM:82
___CS-TR-89-07	___CS-TR-88-03	___5229:TR:86	___5179:TR:85	___5122:TR:84	___5015:TR:82
___CS-TR-89-06	___CS-TR-88-02	___5228:TR:86	___5178:TR:85	___5114:TM:84	___5014:TR:82
___CS-TR-89-05	___CS-TR-88-01	___5227:TR:86	___5174:TR:85	___5112:TR:83	___5012:TM:82
___CS-TR-89-04	___5258:TR:88	___5223:TR:86	___5172:TR:85	___5106:TM:83	___5000:TR:82
___CS-TR-89-03	___5256:TR:87	___5221:TR:86	___5168:TR:84	___5104:TR:83	___4684:TR:82
___CS-TR-89-02	___5253:TR:88	___5220:TR:86	___5165:TR:84	___5094:TR:83	___4655:TR:81
___CS-TR-89-01	___5251:TR:87	___5215:TR:86	___5164:TR:84	___5092:TM:83	___4090-TR-80
___CS-TR-88-22	___5250:TR:87	___5214:TR:86	___5160:TR:84	___5091:TR:83	___3760:TR:80
___CS-TR-88-21	___5249:TR:87	___5212:TR:86	___5158:TR:84	___5090:TR:83	___3759:TR:80
___CS-TR-88-20	___5247:TR:87	___5210:TR:86	___5157:TR:84	___5089:TR:83	___3710:TR:80
___CS-TR-88-19	___5246:TR:87	___5207:TR:86	___5147:TR:84	___5086:TR:83	___3340:TR:79
___CS-TR-88-18	___5244:TR:87	___5205:TR:85	___5143:TR:84	___5082:TR:83	___2276:TM:78
___CS-TR-88-17	___5243:TR:87	___5204:TR:85	___5140:TR:84	___5081:TR:83	___
___CS-TR-88-16	___5242:TR:87	___5202:TR:85	___5139:TR:84	___5074:TR:83	___
___CS-TR-88-15	___5241:TR:87	___5200:TR:85	___5137:TR:84	___5073:TR:83	___
___CS-TR-88-14	___5240:TR:87	___5198:TR:85	___5136:TR:84	___5065:TR:82	___
___CS-TR-88-13	___5239:TR:87	___5197:TR:85	___5135:TR:84	___5054:TM:82	___



SUBMICRON SYSTEMS ARCHITECTURE PROJECT

Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-89-12

31 October 1989

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202; and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-89-12

31 October 1989

Reporting Period: 1 April 1989 – 31 October 1989

Principal Investigator: Charles L. Seitz

Faculty Investigators: K. Mani Chandy
Alain J. Martin
Charles L. Seitz
Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of research activities and results for the seven-month period, 1 April 1989 to 31 October 1989, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- Memoryless Mosaic functional on first silicon (sections 2.1 and 4.9).
- 192-node Symult Series 2010 multicomputer (section 2.2)
- Program Composition (section 3.1)
- Cantor for the Mosaic (section 3.2)
- Testing the asynchronous microprocessor (section 4.1).
- The limits of delay-insensitivity (section 4.2).
- Self-timed mesh-routing chips operate at 65MB/s (section 4.7).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su, Steve Taylor, Tony Wittry

The Mosaic C is an experimental fine-grain multicomputer, currently in development. Each Mosaic node is a single VLSI chip containing a 16-bit processor, a three-dimensional mesh router, a packet interface, 16KB of RAM, and a ROM that holds self-test and bootstrap code. These nodes are arrayed logically and physically in a three-dimensional mesh. We are working toward building a 16K-node (32×32×16) Mosaic prototype, together with the system software and programming tools required to develop application programs.

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small. The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system.

The Mosaic project includes many subtasks, which are listed below together with their current status:

Design, layout, and verification of the single-chip Mosaic node. The Mosaic C chip with 16KB of memory is 9.0mm×7.4mm in a 1.2μm CMOS process, and has 84 pads. Yield characterization indicates that a node with 16KB rather than 8KB of primary memory will increase the chip fabrication cost by less than 30%. Doubling the primary memory at 1.3× the cost for the prototype is a good tradeoff. Additional memory will be particularly helpful for a system that will be used extensively for software development. A substantial economy has been achieved by using TAB rather than conventional packages, so the total fabrication budget has not changed from original estimates.

A "memoryless Mosaic" test chip containing the processor, packet interface, router, clock driver, and central timing and memory arbitration was sent to MOSIS on August 10th to be fabricated in the 1.6μm SCMOS process. (The memory section had been verified earlier.) These chips were returned from fabrication on October 12th, and have been subjected to preliminary tests. Although there are additional tests to perform, this chip appears to operate completely correctly on first silicon, with a yield of 47/50 in the preliminary screening. All processor instructions and the router have been tested; the packet interface is now being tested. The test fixture currently limits speed testing to a clock period of 37ns (27MHz). The chip operates correctly with a clock period of 37ns, except for one case. When an incoming packet

directs the router to switch the packet onto the next dimension, the minimum clock period for correct operation is approximately 65ns. Depending on the nature of the design error, this problem may require a design iteration on the memoryless Mosaic. (See section 4.9 for additional details.)

Internal self-test and bootstrap code. Since the Mosaic C is a programmable computing element, devoting a portion of the bootstrap ROM to self-testing greatly simplifies the logistics of producing these chips in quantity. The bootstrap and self-test code will be tested with EPROM connected to the memoryless Mosaic elements. Additional tests to the channels, which must be accomplished by the fabricator's automatic test equipment, are being written.

Packaging. The packaging design is based on Tape Automated Bonding (TAB) of the chips on small circuit boards. The manufacturing and replacement unit will contain eight nodes in a logical 2×2×2 submesh. These modules have stacking connectors that provide 160 pins on both the top and bottom, and are confined by pressure between motherboards to provide a three-dimensional connection structure that can be disassembled and reassembled for repair. We are currently evaluating suitable connectors.

Cantor runtime system. A Cantor runtime system has been written in Mosaic assembly code, and is now interfaced to the code produced by version 3.0 of the Cantor programming system. Research is underway on runtime algorithms that allow the system to operate robustly in spite of fluctuations in local storage demands. For example, if a local receive queue threatens to overflow, a part of the receive queue is distributed to another node. (See also section 3.2.)

Cantor language, compiler, and application studies. We are now experimenting with version 3.0 of the Cantor language and compiler, which was developed by William C. Athas at the University of Texas at Austin.

Host interfaces and displays. The three-dimensional mesh structure of the Mosaic allows a very large bandwidth around the mesh edges. In order to initiate and interact with computations within the Mosaic, we are designing interfaces between the Mosaic message network and host computers, and between the message network and displays.

A system that will serve both as a prototype of a host interface and as a software development platform is based on eight memoryless Mosaic elements connected to fast, two-ported, external memories. This workstation add-in board will provide an interface that will allow the workstation to monitor the memories of the Mosaic elements during program execution.

In order to provide a high-performance display capability for the Mosaic, we have designed a system that uses one 32×32 plane of a Mosaic as a rendering engine and frame buffer. A detailed design of the video output generator that attaches to one edge of this 32×32 plane has been completed; construction awaits finalization of packaging decisions.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

Symult Systems has delivered additional contributed equipment over the past seven months, with the result that we are now operating a 192-node Symult Series 2010 multicomputer for applications and a 32-node Symult Series 2010 for system development. Utilization of the 192-node system through the Caltech Concurrent Supercomputer Facilities has been at a level of approximately 88% of the available node-hours. These systems run very dependably, and have yet to exhibit a hardware failure.

Copies of the Cosmic Environment system have been distributed on request to 20 additional sites during this period, bringing the total copies distributed directly from the project to nearly 200.

We are implementing a new version of the Cosmic Environment host runtime system, and adding numerous new features to the Reactive Kernel node operating system. The new CE is based internally on reactive-process programming, and will allow a more distributed management of a set of network-connected multicomputers. The extended RK will support global operations across sets of cohort processes, including barrier synchronization, sum, min, max, parallel prefix, and rank. Another extension will be the support of distributed data structures, such as sets and ordered sets. These new features will be implemented at the RK handler level, where the message latency is only a fraction of that at the protected user level. The implementation of these algorithms at the handler level permits global and distributed-data-structure operations in times that do not greatly exceed those of user-level operations dealing with single messages.

Our Caltech project continues to work closely with DARPA-supported Touchstone project at Intel Scientific Computers. Our contributions include the architectural design, message-routing methods and chips, and system software. (See section 3.3 for a summary of the port of RK to the iPSC/2, and section 4.7 for a summary of test results on mesh-routing chips.)

The Cosmic Cubes that were built in our project in 1983 continue to operate reliably. No hard failures were recorded in this seven-month period. The two original Cosmic Cubes have now logged 4.2 million node-hours with only four hard failures; three of these were chip failures in nodes, and one a power-supply failure. A node MTBF in excess of 1,000,000 hours is probable based on this reliability experience.

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

3. Concurrent Computation

3.1 Program Composition

K. Mani Chandy, Steve Taylor

This research investigates the use of program composition as a method of developing concurrent programs. The goal is to develop a theory, a notation, and an implementation of program composition operators so that programs can be developed by putting smaller programs together to get larger ones. The compositional approach to programming was described in the previous semiannual technical report. New components of this work are:

1. A primitive set of composition operators (and not merely sequential or functional composition) has been implemented, and a proof theory has been developed for this set of operators.
2. The researchers believe that in each application area there are a few problem-solving paradigms or "templates," and that, formally, these templates are user-defined composition operators. Thus, the notation allows user-defined composition operators.
3. The notation is intended to execute on both shared-memory and message-passing concurrent computers, without modification. A fragment of the notation has been implemented on the Connection Machine by Professor Rajive Bagrodia at UCLA.
4. The theory incorporates functional programming ideas, and extends it to problems that are not functional. (Most reactive systems are nondeterministic, and nonfunctional.)
5. The researchers have been working with computational fluid dynamicists and biologists to identify problem-solving paradigms in these disciplines, and to evaluate whether the compositional approach is effective in these areas.

The theory of program composition has been developed, and a prototype implementation in Strand has been completed. Discussions with Caltech faculty in Applied Math and Biology have provided initial test cases. Discussions with researchers at Aerospace Corporation have allowed an evaluation of program composition for tracking and trajectory-computation applications, and have led to initial joint research in these applications.

3.2 Cantor for the Mosaic

Nanette J. Boden, Chuck Seitz

With the Cantor version 3.0 compiler and interpreter in place, we are beginning to translate a representative subset of our library of Cantor application programs into the new version. The purpose of this exercise is twofold: We maintain a library of programs for demonstrations, and we continue the process of evaluating the impact

of new language features on application programming. The aspects of the Cantor 3.0 that have the most impact on programming are the incorporation of functions and the introduction of message discretion.

As usual in the development of programming systems, the introduction of new capabilities at one level of the system imposes new requirements at other levels. In the case of the new features of Cantor 3.0, the introduction of message discretion raises the specter of violating the guarantee of message consumption. If a process is waiting for the arrival of a particular message, messages received in the interim must be buffered. Since the resources *of a node* are quite limited, physical space may not be available for the awaited message to be received. Since infinite queueing is theoretically required, we are investigating engineering solutions that use the resources *of the entire machine,* and potentially of secondary memory, to approximate infinite queues.

In addition to implementing runtime support for new language features, we are investigating solutions to other problems that became apparent during the development of the Mosaic runtime system. In this first version, we made simplifying assumptions to minimize both the size and complexity of the runtime support. Two of the assumptions that must be seriously addressed in future versions of the runtime system are: (1) if an available reference value exists for the creation of a new process on a remote node, then enough resources exist on that node for the new process; and (2) the code for each process resides on every node. These assumptions are clearly unrealistic for the types of memory-intensive computations that we seek to perform. Currently, we are devising and evaluating schemes for process placement that do not assume available resources on the remote node. We are also devising schemes for code partitioning that will maximize the amount of memory available for processes, while not introducing excessive overhead for acquiring necessary copies of process code.

3.3 The Cosmic Environment and Reactive Kernel

Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Wen-King Su

A joint effort with Intel to port the Reactive Kernel to run as the native node operating system on the iPSC/2 has successfully achieved its first milestone. Our longer-term goal is to run an enhanced version of RK on a future Intel multicomputer that is based on the Intel i860 processor.

The port of the Inner Kernel of RK and of the system-handler layer was performed in an intensive effort over a two-week period by Jakov Seizovic, RK's original author, and was upgraded to include a preliminary user-process handler by Bill Bain of Intel during the following two weeks. The fine-tuning of the message performance took another week. This port has shown once again that the modular structure of RK provides for simple porting and simplifies debugging, especially in the early phases of the port. This preliminary version of RK outperformed

the Intel NX operating system by about a factor of two in message latency, and achieved equivalent message bandwidth. We have subsequently increased the message bandwidth while providing proper fragmentation and reassembly of long messages, which increases the fairness of access to the message network. The completion of this port is expected to be performed principally by Intel within the next two months.

RK has gotten somewhat ahead of the Cosmic Environment system in its use of a layered reactive-process structure. A new version of CE has been designed, and is currently being written.

3.4 Hybrid Distributed Discrete-Event Simulators

Wen-King Su, Chuck Seitz

Two hybrid distributed simulators have been written, and their performance results are included in the PhD thesis: "Reactive-Process Programming and Distributed Discrete-Event Simulation," [Caltech-CS-TR-89-11].

In a distributed discrete-event simulation, the simulation subject is divided into a number of smaller elements. The elements are distributed over a multicomputer or a multiprocessor, and are simulated concurrently. In a conservative simulator, null messages are necessary for the progress of a circuit of idling elements. In the framework of the Chandy-Misra-Bryant algorithm, elements are simulated independently, as if each element is located on a separate node. While this framework will achieve good performance on a fine-grain multicomputer, the volume of null messages is an unnecessary burden for a medium-grain multicomputer, in which many elements share the same node. When nodes are few, the CMB simulator does worse than a sequential simulator.

The goal of the hybrid simulators is to eliminate intra-node null messages by combining elements on the same node into a single macro-element. In the hybrid-1 simulator, macro-elements are simulated internally by a conventional sequential simulator. Hybrid-1 reduces intra-node messages by eliminating all intra-node null messages. It also reduces inter-node messages by synchronizing all element outputs in a macro-element. The result is a simulator that equals a sequential simulator on a single node and shows a speedup when more nodes are used, regardless of element placement. However, the amount of speedup is limited because some concurrency is lost to the strict synchronization. In hybrid-2, macro-elements are simulated by a combination of CMB and sequential simulators. Elements are constantly moved between the two modes as they become blocked or unblocked. Since an element can progress as far as its inputs allow, the hybrid-2 can attain the full CMB speedup when many nodes are used. However, since element outputs are not synchronized in each macro-element, hybrid-2 is sensitive to element placement.

3.5 CONCISE*

Sven Mattisson, Lena Peterson, Chuck Seitz

The concurrent circuit-simulation program, CONCISE, originally used waveform relaxation in conjunction with Jacobi iterations. This method gives high concurrency, but other iterative methods have better convergence performance. These other methods do not, however, offer the same concurrency as the Jacobi method. Thus, we have concentrated recently on developing combinational methods that retain the concurrency properties of the Jacobi iterations while improving convergence. CONCISE has been enhanced to exploit circuit-node coupling. The strongly coupled nodes are solved in a block with a direct method; thus, convergence is improved.

The waveform relaxation method has also been augmented with multicolored Gauss-Seidel iterations. Normally, Gauss-Seidel iterations rely on the equations being solved in sequence. However, by coloring the circuit graph it is possible to find an ordering that gives high concurrency. All equations with one color can be solved in parallel, and typically only three to five colors are needed for a circuit to yield high concurrency.

A special version of CONCISE was written to evaluate Jacobian matrix coefficients concurrently, while using a single-rate integration method for each subsystem. This version is now about to be incorporated in the standard version.

A plotting program, communicating with CONCISE via messages, has been developed. This program displays selected waveforms as they are computed.

CONCISE was given a thorough workout over the summer performing simulations on a 64-node Symult 2010 of 4000-transistor sections of the FMRC2.1 self-timed mesh-routing chips. These studies were part of characterizing the process-dependence of the FRMC2.1 design.

CONCISE is written in C using the CE/RK functions, and now runs on Sun, Sequent, Macintosh II (A/UX), Intel iPSC/1, Intel iPSC/2, and Symult Series 2010 computers.

3.6 A C-Based Concurrent Programming Language For Multicomputers

Marcel van der Goot, Alain Martin

We are defining and implementing a concurrent programming language for message-passing multicomputers. Since the main difference between multicomputers and sequential machines is the possibility of concurrency, we have concentrated in our language design on adding concurrency without redefining the complete computation model. In particular, since most of our programming experience is

* This segment of our research is a joint project with the Applied Electronics Department of the University of Lund, Sweden.

with using imperative sequential languages, we have chosen one such language, C, as the basis for our work. C matches well with our desire to design a language that is compact but nevertheless useful for writing "real" application programs.

In our model, a computation consists of a set of independently executing sequential processes, plus a set of message-buffers (channels) connecting pairs of processes. Processes and channels together form the so-called computation graph, which can vary dynamically during the computation. A process is a short sequential (C) program that can exchange data with its environment by sending or receiving messages. A process typically has about the same size as a function; such a fine grain size makes the language applicable to a large range of multicomputers.

We finished a preliminary implementation of a somewhat restricted version of the language earlier this summer. In that implementation, a concurrent program is compiled into a single UNIX process that is executed on a Sun workstation. Currently we are working on a compiler for the complete language, which we hope to have running in December or January.

4. VLSI Design

4.1 Testing of the Asynchronous Microprocessor

Steve Burns, Tony Lee, Dražen Borković, Pieter Hazewindus, Alain Martin

The Asynchronous Microprocessor, described in the previous semiannual technical report, has since been thoroughly tested. Chips fabricated at a 2μm feature size functioned as intended over a wide range of power supply voltages, temperatures, and delays of the external memories. The chips fabricated at 1.6μm, while functioning correctly at certain voltages, temperatures, and delays, failed for many values of these external parameters. After a detailed analysis, we concluded that all the high-level transformations were performed correctly. The problem, instead, occurred in the final phase of the compilation, the transformation from production rules into networks of CMOS gates. In particular, the values of some isochronic forks change too slowly, allowing different gates to interpret the digital value inconsistently. These forks were located and the circuits were modified to correct the problem. A corrected 1.6μm version of the microprocessor is expected back from MOSIS fabrication on December 1st.

4.2 The Limitations to Delay-Insensitivity in Asynchronous Circuits

Alain Martin

Once it was established that the problem in the 1.6μm version of the microprocessor was caused by a malfunctioning of an isochronic fork for certain values of the external parameters, the question of whether isochronic forks are necessary needed to be answered.

An isochronic fork is used to distribute a variable to several points of the circuit as input of several gates. In the discrete model, it is assumed that the different copies of the variable have the same values at all times. For this assumption to be valid, the following timing requirement has to be fulfilled. A change on the input of a fork causes the different outputs to change asynchronously. However, the "transition delays" on the different outputs of an isochronic fork must be similar enough in length that once a change on one of the outputs of the fork has caused another gate to fire, one may conclude that the changes on all the outputs have completed.

Since the definition of isochronic forks violates the delay-insensitivity assumption, and since all efforts to design entirely delay-insensitive circuits have been fruitless, we started to suspect that the class of circuits that are entirely delay-insensitive could be very limited. Indeed, we have been able to prove that an entirely delay-insensitive circuit can contain only C-elements, hence settling an important open question in the theory of asynchronous circuit design, and vindicating the compromise to delay-insensitivity implied by the use of isochronic forks.

4.3 Tools for Performance Evaluation of Self-timed Circuits

Steve Burns, Alain Martin

The compilation method has, in the past, been mostly concerned with correctness, not efficiency. With the design of the microprocessor, high performance has become a major concern. Two separate analysis tools have been developed in order to determine the speed at which self-timed circuits operate.

The first tool is a simple event-driven simulator that takes, as input, extracted circuit layout. Timing analysis is based on the τ-model. Good agreement has been found between the timing information produced by the simulator and actual results obtained from the fabricated chips. The simulator itself is quite efficient, even for large circuits; simulation of a single instruction of the microprocessor takes less than a second.

The second tool allows the comparison of various methods of handshaking without actually constructing and then simulating the circuit. The fundamental sequencing between actions can be determined, in many important cases, by a static analysis of a high-level description of the program. The necessary analysis involves solution of a finite linear optimization problem. For small problems, it can be solved by the enumeration of all the cycles in a so-called "constraint" graph. A PROLOG program has been constructed that performs this analysis.

4.4 Cache Memory for an Asynchronous Microprocessor

José A. Tierno, Alain Martin

The design of a direct-mapped instruction cache for an asynchronous microprocessor is underway. The cache is completely self-timed, both the control part and the RAM array. The objective is to make the design suitable for on-chip implementation as part of the processor pipeline.

4.5 Self-Timed Circuits in GaAs

José A. Tierno, Alain Martin

Experimentation is being done on new transistor configurations for digital circuits implemented in Enhancement/Depletion mode MESFET GaAs technology. The main characteristics of these configurations are increased noise margins, reduced input load, and slightly faster gate delays than conventional DCFL (direct-coupled FET logic) and SBFL (super buffered fet logic) technology. Extensive experimentation has been done using SPICE for simulations, and two chips have been sent for fabrication to test some basic circuits.

4.6 Testing Self-Timed Circuits

Pieter Hazewindus, Alain Martin

We are continuing our investigation into the testability of self-timed circuits. Previously we tried to construct a set of circuit elements with which any program could be implemented and for which all faults would be testable. This goal seems unattainable: We have found that – with one exception – for any isochronic fork there is a corresponding fault that is not testable. As we have shown that most circuits require isochronic forks, the range of circuits without untestable faults is very limited. Hence, without additional circuitry or additional scan points, most circuits will have untestable faults.

To increase the fault coverage it is possible to add a test structure, thus connecting all state-holding elements in a queue and thereby reducing the problem of testing a sequential circuit to that of testing a combinational one. Test vectors are put into the queue, while the results are taken out, similar to scan-type designs for synchronous circuits. We speculate that with appropriate conditions on the combinational logic, all faults are testable this way; however, for our current design style, such a queue would be expensive in area, as the number of state-holding elements is much larger than the number of latches in a typical synchronous design. We are investigating ways to reduce the number of state-holding elements in the queue while maintaining the complete testability.

4.7 Fast Self-Timed Mesh Routing Chips

Chuck Seitz

The FMRC2.1 mesh-routing chips have now been thoroughly characterized by Intel, and have been shown to operate at a channel rate of 65MB/s. However, testing at Intel also discovered a failure mode that occurs when several channels operate concurrently. This failure was traced to collapse of the internal power supply under these demanding conditions; thus, it is properly a failure of the packaging rather than of the chip design.

This 132-pin chip devotes the 20 lowest-inductance PGA-package pins to Vdd and GND, but either a better package or twice as many Vdd and GND pins are required. Experiments with a number of alternative packages are now underway, and have involved producing a complete set of test vectors for automatically testing MRC chips.

The design and layout of two other versions of the FMRC is now underway. One of these versions is designed to minimize latency by using relatively few internal FIFO stages. Multicomputer applications benefit from the internal FIFOs, which reduce blocking contention, but the tradeoff between throughput and latency is different for multiprocessor applications.

Samples of tested FMRC2.1 chips have been provided in recent months to CMU,

MCC, and MIT, in addition to those samples provided earlier to Intel Scientific Computers and Symult Systems.

4.8 Implementing Adaptive Routing in Multicomputer Networks

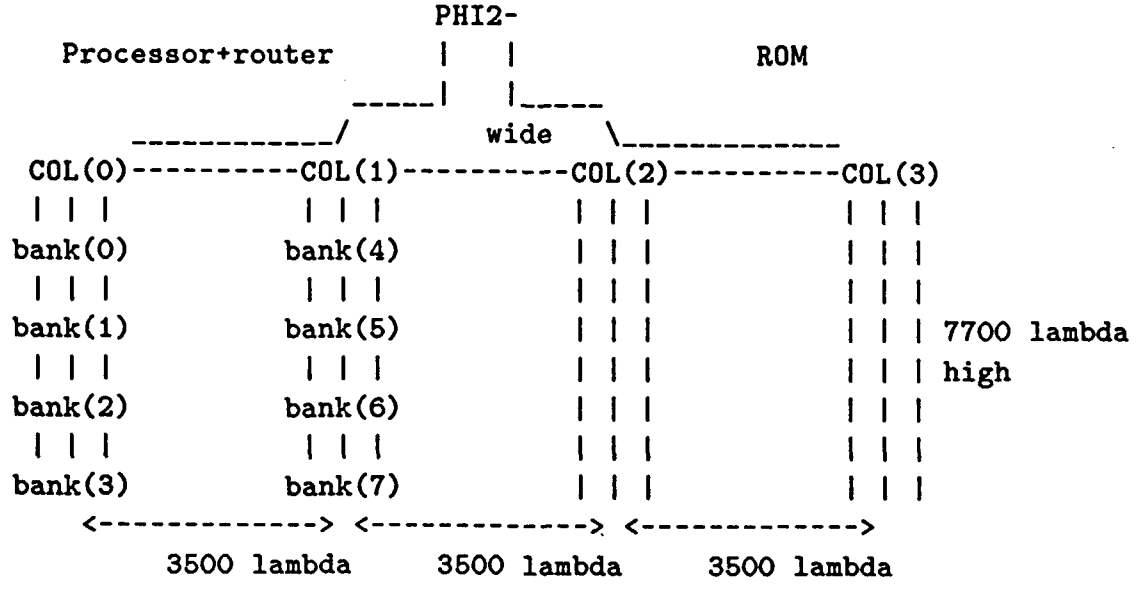
Mike Pertel, Chuck Seitz

We are investigating those performance enhancements in multicomputer routing that are achievable through practical adaptive routing strategies. Earlier work has demonstrated the potential of multipath routing; our current objective is the realization of that potential. The initial phase of this work is the comparison of various specific routing algorithms on the basis of low-latency throughput, fault tolerance, and traffic diffusion. The algorithm found to exhibit the best performance under detailed network simulation will be implemented as a VLSI circuit to replace the current Mosaic router.

4.9 Mosaic C Chips

Jakov Seizovic, Chuck Seitz, Don Speck, Wen-King Su, Tony Wittry

The full Mosaic element, a 9.0mm×7.4mm chip in 1.2μm SCMOS technology, introduced us to a number of difficulties in the design of chips that exhibit both high complexity (≈700K transistors) and fairly high clock rate (40MHz). The clock lines cannot be run in minimum-width metal without compromising performance. In this design, the memory contributes the largest part of the clock load, and the combination of capacitive load and line resistance require that the clock lines be run across the backbone of the chip in ≈10μm-wide metal. The critical clock line happened to be $\phi 2'$, which is distributed from the clock driver to the memory section in the following pattern:



Analytical and simulation results for this situation were nearly identical. For simulation simplicity, the set of three, parallel, minimum-width wires in each vertical bundle was treated as one wide wire. Only the left half of the distribution network was simulated with SPICE. The result of these simulations confirmed the following area-energy-period optimums: (1) a clock driver of 700sq n-channel + 1050sq p-channel per half of the RAM per phase, and (2) 15λ-wide distribution wires across the top for each phase.

The memory and router sections of the Mosaic were fabricated and tested more than a year ago. To test the remaining parts of the full Mosaic element and their ability to work together, the processor, packet interface, router, and clock driver were integrated onto a "memoryless Mosaic" test chip that was sent to MOSIS on August 10th. This test chip was a major milestone for us. A number of improvements in the processor instruction set and an increase in the channel bandwidth created some design imbalances that required a substantial amount of rethinking of the memory arbitration and packet interface.

The maximum combined data bandwidth of the receive and send parts of the packet interface (PI) is equal to 50% of the total memory bandwidth, which is one 16-bit read or write each clock period (25ns). The original design specifications for the PI included the assumption that there would be enough spare memory cycles so that the PI would not need to request access to the data bus; it would instead use otherwise unused cycles for message transfer from/into the network.

The increased efficiency of the processor microcode invalidated this assumption, and the design of the memory-bus arbitration unit and the PI had to be modified to comply with the new specifications. Eight-word buffers were added between the memory and the sending part of the packet interface, and between the receiving part of the packet interface and the memory. The signals generated by the sender and the receiver part of PI to request access to the memory bus include "hysteresis": Depending on the amount of space available in the buffers, the PI may either steal unused cycles or request exclusive use of the memory. This scheme allows the data transfer between memory and buffers to occur in bursts, rather than imposing the bus arbitration overhead on every PI memory access.

The new design provides for the data transfer from/into the network at the full network bandwidth regardless of the instruction sequence being executed. This feature was achieved with a fairly modest increase in complexity: an increase in buffering space from two to sixteen words, and an additional state machine to handle the bus-request logic.

We are currently in the process of testing the memoryless Mosaic chips that were returned from MOSIS fabrication on October 12th. So far, the chips appear to function correctly. All processor instructions and router functions operate correctly, but there is evidently a slow path in the router. We are investigating this problem.

The Limitations to Delay-Insensitivity in Asynchronous Circuits

Alain J. Martin
Department of Computer Science
California Institute of Technology
Pasadena CA 91125, USA

Asynchronous techniques—i.e. techniques that do not use clocks to implement sequencing—are currently attracting considerable interest for digital VLSI circuit design, in particular when the circuits produced are *delay-insensitive*. A digital circuit is delay-insensitive (d.i.) when its correct operation is independent of the delays in operators and in the wires connecting the operators, except that the delays are finite and non-negative.

In this paper, we characterize the class of circuits that are entirely delay-insensitive, and we show that this class is surprisingly limited: Practically all circuits of interest fall outside the class since circuits inside the class may contain only C-elements as multi-input operators.

1. Circuits as Networks of Gates

A d.i. circuit is a network of logical operators, or *gates*. A gate has one or more Boolean inputs and one Boolean output. A gate represents a Boolean function: For constant values of the inputs, the output takes a value that is defined by a Boolean function of the inputs, and possibly the current value of the output. The state of the circuit is entirely characterized by the set of input and output variables of the gates.

We assume that all circuits are *closed*: Each variable is the input of a gate and the output of a gate. (We shall see that we can ignore self-loops and postulate that a variable is shared by exactly two different gates.) An open circuit is transformed into a closed one by representing the environment of the circuit as gates.

Definitions and Notations. *An execution of a simple assignment is called a transition. The result of a transition of type* $x\uparrow$ *is the postcondition* x*; the result of a transition of type* $x\downarrow$ *is the postcondition* $\neg x$*. The simple assignments* $x := true$ *and* $x := false$ *are denoted by* $x\uparrow$ *and* $x\downarrow$*, respectively.*

A gate with output variable z is defined by the two *production rules* (p.r.'s):

$$B_u \mapsto z\uparrow$$
$$B_d \mapsto z\downarrow$$

where B_u is the condition on the input variables for a transition $z\uparrow$ to take place, and B_d is the condition on the input variables for a transition $z\downarrow$ to take place—B_u and B_d are called the *guards* of the p.r.'s. The two production rules of a gate have to fulfil the *non-interference* requirement.

Non-interference. $\neg B_u \vee \neg B_d$ *is invariantly true.*

The *result of a production rule* is the result of the transition caused by the execution of a production rule.

All production rules with a true guard are executed concurrently. The execution of a production rule is considered correctly terminated when the result holds. The execution of a p.r. correctly terminates except if the guard is falsified before the result holds. In that case, the net-effect of the execution is undefined. We therefore add a semantic requirement (to be proved invariantly true): *stability* of a guard in a computation.

Stability. *The guard of a production rule is stable in a computation when it is falsified only in states where the result of the production rule holds.*

We exclude *self-invalidating* production rules. A rule with guard g and result r is self-invalidating if $r \Rightarrow \neg g$, like, for example, the rules $x \mapsto x\downarrow$ and $\neg x \mapsto x\uparrow$.

The execution of a p.r. in a state where the result holds is called *vacuous*, and is called *effective* otherwise. From the definition of the execution of a p.r., the vacuous execution of a p.r. is equivalent to a *skip*. Consequently, it is always possible to modify the guard of a p.r. so that it does not contain the output variable of the gate. (Left as an exercise for the reader.) Hence, we can eliminate self-loops, i.e., variables that are input and output of the same gate. In the sequel, unless specified otherwise, an execution of a p.r. means an effective execution.

2. Wires, Forks, and Multiple-Output Gates

A priori, a wire with input x and output y is the gate defined by the p.r.'s $x \mapsto y\uparrow$ and $\neg x \mapsto y\downarrow$. But the composition of any gate, *including a wire*, with a wire is the gate itself with one of its variables renamed. Hence, we can add an arbitrary number of wire gates to a circuit definition without actually changing the circuit. In order to have a unique network of gates for each circuit, we exclude the wire from the repertoire of gates: A wire is just a renaming mechanism for variables.

We also exclude the *fork* from the repertoire of gates. A fork has one input and at least two outputs. The fork f with input x and outputs y and z is defined by the two p.r.'s $x \mapsto y\uparrow, z\uparrow$ and $\neg x \mapsto y\downarrow, z\downarrow$. The generalization to an arbitrary number of outputs is obvious. The gate

$$B_u \mapsto x\uparrow$$
$$B_d \mapsto x\downarrow$$

composed with fork f is equivalent to the gate with outputs y and z

$$B_u \mapsto y\uparrow, z\uparrow$$
$$B_d \mapsto y\downarrow, z\downarrow$$

Hence, the fork is just a mechanism for replicating the outputs of a gate and for defining gates with an arbitrary number of outputs. But gates defined in this way have an important restriction: *The effective execution of a production rule of a gate contains an effective transition on each output of the gate.*

The only restriction on the class of circuits considered that these definitions and conventions introduce is the exclusion of arbitration devices. They do not restrict the delay-insensitivity assumption.

3. Partial Order of Transitions

The specification of a circuit defines a partial order of actions taken from a repertoire of commands. In order to assert that a circuit implements a specification, we relate this partial order to some other order relation among transitions of a circuit. And we will say that a circuit implements a specification when the partial order of transitions in each computation of the circuit contains the specification partial order in a way that we will explain later.

Consider an effective execution of a p.r. with term C of the guard true, and let t be the transition of this execution. (We assume that the guard is in disjunctive normal form, i.e., it is either a *literal*, or a *term*, or a disjunction of terms. A literal is a variable or its negation, and a term is a conjunction of literals.)

We attach to C a set T of transitions in the following way. Each literal of C uniquely defines a transition: The literal x is the result of a transition of type $x\uparrow$; the literal $\neg x$ is the result of a transition of type $x\downarrow$. (The initialization of a variable is also considered a transition.) *By definition, we say that transition t is a successor of each transition of T.*

From the *successor* relation, we can now construct a relation $\prec$ which is a pre-order, i.e., it is transitive and anti-reflexive.

Transitivity For any two transitions $t1$ and $t2$, we say that $t1 \prec t2$ when $t2$ is a *successor* of $t1$, or there exists a transition $t3$ such that $t1 \prec t3$ and $t3 \prec t2$.

Anti-reflexivity $t \prec t$ holds for no transition t.

Anti-reflexivity is satisfied if, for each ring of gates in the circuit, there is always at least one p.r. whose guard is true and whose result is false—the ring "oscillates." Anti-reflexivity excludes rings of gates that are used to maintain constant values of variables, like in cross-coupled device constructions of storage elements. We therefore assume that the storage elements are parts of "perfect wires," so to speak, which keep the value of a variable until the next transition on the variable.

Once we have the pre-order relation $\prec$, we construct the partial order $\preceq$ by defining $t1 \preceq t2$ to mean $t1 \prec t2$ or $t1 = t2$.

Definition. *A chain from a to b is a finite, non-empty set $\{t_i, 0 \leq i < n\}$ of transitions such that $t_0 = a$, $t_n = b$, and for all i, $0 < i < n$, t_i is a successor of t_{i-1}. By construction, $a \preceq b$ means that there is a chain from a to b.*

If $a \prec b$, we will somtimes say that b *follows* a.

4. Implementation of Stability

Consider again an execution of p.r. with guard B and transition t. Either B is never falsified once it holds, but then t is the last transition on the variable involved—we say that the transition is *final.* Or B is falsified after a finite number of transitions following t. In that case, in order to implement stability, we have to see to it that t is completed before B is falsified.

For all transitions i that falsify B, we have to guarantee $t \prec i$. Hence, by definition of the order relation, there must be a transition s such that s is a *successor* of t, and $s \preceq i$. We say that s *acknowledges* t.

Acknowledgment Theorem. *In a d.i. circuit, each non-final transition t has a successor transition.*

By construction of multiple-output gates, we have the

Corollary. *In a d.i. circuit, a non-final transition on an input of a gate has a successor transition on each output of the gate.*

5. The Unique-Successor-Set Criterion

Later on, we shall give a simple criterion to decide whether a given circuit—a network of gates—is delay-insensitive. But such a criterion does not tell us whether there exists a d.i. circuit for a given specification. We shall therefore formulate a more general theorem which characterizes the partially ordered sequences of transitions that admit a d.i. implementation. This criterion enables us to decide that a program does not have a d.i. implementation without having to construct a circuit.

A *computation* is a partially ordered sequence of transitions corresponding to a possible execution of a circuit. It is finite if the computation terminates, and infinite otherwise.

Successor Set. *In a computation, the successor set of a transition t is the set of variables x such that a transition tx on x is a successor of t.*

Unique-Successor-Set Property. *A computation has the unique-successor-set (USS) property when all non-final transitions on the same variable have the same successor set. A set of computations has the USS property when all non-final transitions on the same variable have the same successor set in all computations of the set.*

Unique-Successor-Set Theorem. *A set of computations of a d.i. circuit has the USS property.*

Proof. From the definition of the successor of a transition and the corollary, the successor set of a non-final transition on a variable, say, y, is the set of output variables of the gate of which y is an input.

Since this gate is uniquely defined by the circuit topology, the successor set is unique for all transitions on y in all computations corresponding to an execution of the circuit. □

Although the Unique Successor Set Theorem is a direct consequence of the Acknowledgment Theorem, its formulation in terms of computations instead of gates makes it possible to lift the result from the implementation level to the specification level. We assume that whatever specification notation is used—programs, traces, regular expressions, temporal logic, etc.—it is possible to derive certain properties of the partial ordering of actions involved from the specification. Hence, in the sequel, a specification means a partially ordered sequence of actions taken from some repertoire of commands.

Since the partially ordered sequence of actions defining the specification is a projection of the sequence of actions implementing it, we shall investigate whether the USS property is maintained by projection.

Definition. *Given a computation c on a set V of variables, the projection of c on a subset W of V is the computation derived from c by removing all transitions on variables of $V \setminus W$ from the chains of c. The projection of a set of computations is the set obtained by projecting each element of the original set.*

Projection Theorem. *If a set of computations has the USS property, then its projection on a subset of variables has the USS property.*

Proof. By definition, the projection of a set of computations on W can be obtained by removing the elements of $V \setminus W$ one for one from all chains of each computation of the set. We prove the theorem by showing that removing all transitions on one variable, say, w, maintains the USS property of the set.

Let x be another variable, and let X be the USS of (all transitions on) x in all computations of the set. Either w does not belong to X and X is left unchanged by the transformation. Or w is removed from X. But then, for each transition tx on x, the successor set of the transition on w that follows tx has to be added to the successor set of tx. Since all transitions on w have the same successor set in all computations of the set, the new X is the same for all transitions and all computations of the set. □

EXAMPLE

The cyclic program $*[X;Y]$ where X and Y are communication commands is called a *one-place buffer*. It is a basic building block of asynchronous circuit design. With a four-phase handshaking protocol for implementating the communications, an expansion of the program is in terms of elementary variables is:

$$*[[xi]; xo\uparrow; [\neg xi]; xo\downarrow; yo\uparrow; [yi]; yo\downarrow; [\neg yi]]$$

where xi and yi are the input variables, and xo and yo are the output variables. The command $[B]$ is a shorthand notation for $[B \rightarrow skip]$, and can be informally described as "wait until B holds".

The environment of the circuit can be simply modeled as the two programs:

$$*[xi\uparrow; [xo]; xi\downarrow; [\neg xo]]$$
$$*[[yo]; yi\uparrow; [\neg yo]; yi\downarrow].$$

The three programs are concurrent. Now observe that the projection of an infinite computation on the input variables of the first program gives the infinite computation described by the program

$$*[xi\uparrow; xi\downarrow; yi\uparrow; yi\downarrow].$$

Obviously, this infinite computation does not have the USS property and, therefore, the closed circuit implementing the three programs is not d.i.. But the two environment programs can be implemented with an inverter and a wire, which are d.i circuits. Hence, a circuit implementing this version of the one-place buffer is not d.i.. □

6. Specifications and the USS Property

The Projection Theorem is very useful because we can also define when a specification has the USS property, so that if a specification does not have the property, we can immediately conclude that there exists no d.i. implementation of the specification. The projection from implementation to specification occurs as follows.

Given is an arbitrary command (or statement) S; S can be of any kind: assignment, communication, procedure call, transition of a finite-state machine, etc. We make the—in theory slightly restrictive—assumption that an elementary variable can be uniquely identified with each command of the repertoire, i.e., the transitions on the variable occur in the executions of the command only, and each execution of the command contains a transition on the variable. (This assumption is needed for the specification theorem only.)

Consider then a specification implying a certain partial order of actions on a given repertoire of commands X, Y, Z, etc. This partial order—which we now call the *specification*—implies the same partial order on a set of transitions on the elementary variables x, y, z, etc., that can be uniquely attached to the commands.

Hence, the specification defines a projection of the computation on the set of variables $\{x, y, z, ...\}$. According to the Projection Theorem, we can then formulate the following

Specification Theorem. *If the specification of a circuit does not have the USS property, the circuit has no d.i. implementation.*

EXAMPLES The following examples, which we give without proofs, show how limited is the class of programs that admit a d.i. implementation. (In the examples, all commands are different from the empty command *skip*.) We assume that the

semantics of the program notation is clear enough that we can identify the programs with the partial order of actions they represent.

- Let $P \equiv *[S_1; S_2; \ldots S_n]$, and assume that there is no equivalent program

$$*[S_1; S_2; \ldots S_k]$$

with $k < n$. (We say that P is a minimal representation. For instance, $*[X; X]$ is not minimal since $*[X]$ is an equivalent program.)

Then P has the USS property only if $S_i \neq S_j$ for $i \neq j$. (That the condition is not sufficient is shown by the previous example.) Hence, the "modulo-2 counter" $*[X; X; Y]$ and all other "modulo-k counters" have no d.i. implementation.

- The program $*[S_1; [B_1 \rightarrow S_2 [] B_2 \rightarrow S_3]; S_4]$, with $S_2 \neq S_3$, does not have the USS property. Hence, there is no d.i. circuit implementing such a selection command.

□

7. Gate characterization of d.i. circuits

Definition. *An n-input gate in which B_u is the conjunction of the n input variables and B_d is the conjunction of the negations of the n input variables is called an n-input C-element. A gate derived from a C-element by negating one or more literals in B_u or B_d is also a C-element.*

The Muller-C element is a two-input C-element according to our definition. A one-input C-element reduces to either a wire or an inverter.

C-element Theorem. *If each computation of a d.i. circuit contains at least 3 transitions on each variable, the circuit comprises only C-elements, or gates that can be replaced with C-elements.*

Proof. Let x be an arbitrary variable of the circuit; x is the input of gate g with output z. We shall prove that g can be implemented as a C-element.

We consider an arbitrary computation of the circuit. First, observe that because of the non-interference, all transitions on the same variable are totally ordered. And because all transitions are effective, upgoing and downgoing transitions on the same variable alternate.

Since the circuit contains at least 3 (effective) transitions on each variable, at least one transition of type $x\uparrow$ is followed by a transition of type $x\downarrow$. And at least one transition of type $x\downarrow$ is followed by a transition of type $x\uparrow$.

Let $t1$ be a transition of type $x\uparrow$ and $t2$ the transition of type $x\downarrow$ following it. For the guard of the p.r. of $t1$ to be stable, there must be a transition tz on z such that $t1 \prec tz \prec t2$. We also know that tz is a successor of $t1$.

By the USS theorem and the Projection theorem, there is exactly one transition tz on z such that $t1 \prec tz \prec t2$. By the same argument, there is exactly one

transition on z between a transition of type $x\downarrow$ and the transition of type $x\uparrow$ following it.

Without loss of generality, assume that the first transition on x is of type $x\uparrow$ and the first transition on z is of type $z\uparrow$. Then, because of the alternation of upgoing and downgoing transitions on each variable, each transition of type $z\uparrow$ is the successor of a transition of type $x\uparrow$. And each transition of type $z\downarrow$ is the successor of a transition of type $x\downarrow$.

By definition of the successor relation, all terms of guard B_u of g contain x. Hence, B_u is of the form $x \wedge C_u$, where C_u does not contain x. Symmetrically, guard B_d of g is of the form $\neg x \wedge C_d$, where C_d does not contain x. Since this property of B_u and B_d holds for each input of g, g is a C-element or can be replaced with a C-element. □

8. For Whom the Bell Tolls?

Are these results tolling the bell of d.i. design? Actually, not. At worst, they may slightly embarass those researchers who claim to have a design method for entirely d.i. circuits. At best, they vindicate the compromises to delay-insensitivity adopted by several asynchronous design methods.

The compromise I have introduced is that of *isochronic forks*. In an isochronic fork, the transitions on *all* outputs of the fork are completed when a transition on *one* output has been acknowledged. Hence, some transitions on some outputs of an isochronic fork need not be acknowledged, and thus the Acknowledgment Theorem does not always hold.

The extension of a standard repertoire of d.i. gates with isochronic forks is sufficient to construct any circuit of interest. I believe it is the weakest possible extension in the sense that any other compromise includes isochronic forks.

9. Acknowledgments

The formulation of the C-element Theorem in terms of three transitions on each variable is due to a suggestion from Pieter Hazewindus. Acknowledgment is also due to Steve Burns, Peter Hofstee, Marcel van der Goot, Tony Lee, and José Tierno for their comments and criticisms. The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

A Framework for Adaptive Routing in Multicomputer Networks

John Y. Ngai and Charles L. Seitz
Department of Computer Science
California Institute of Technology*

Introduction

Message-passing concurrent computers, also known as *multicomputers*, such as the Caltech Cosmic Cube [1] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes [2]. The communication networks of the second-generation machines, such as the Symult Series 2010 and the Intel iPSC2, employ an *oblivious* wormhole routing technique [3,4] that guarantees deadlock freedom. The network performance of this highly evolved oblivious technique has reached a limit of being capable of delivering, under random traffic, a *stable* maximum sustained throughput of $\approx$ 45 to 50% of the limit set by the network bisection bandwidth. Further improvements on these networks will require an *adaptive* utilization of available network bandwidth to diffuse local congestions.

In an adaptive multipath routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. It is expected that such an adaptive control can increase the throughput capability towards the bisection bandwidth limit, while maintaining a reasonable network latency. While the potential gain in throughput is at most only a factor of 2 under random traffic, the adaptive approach offers additional advantages, such as the ability to diffuse local congestions in unbalanced traffic, and the potential to exploit inherent path redundancy in richly connected networks to perform *fault-tolerant* routing. The rest of this paper consists of an examination of the various feasibility issues and results concerning the adaptive approach studied by the authors. A much more detailed exposition, including results on performance modeling and fault-tolerant routing, can be found in [5].

*The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

Published in SPAA '89, June 1989,

The Adaptive Cut-Through Model

It is clear that in order for the adaptive multipath scheme to compete favorably with the existing oblivious wormhole technique, it must employ a switching technique akin to *virtual cut-through* [6]. In cut-through switching and its blocking variant, which is used in oblivious wormhole routing, a packet is forwarded immediately upon receipt of enough header information to make a routing decision. The result is a dramatic reduction in the network latency over the conventional store-and-forward switching technique under light to moderate traffic. We now describe a simple cut-through switching model that provides the context for the discussion of issues involved in performing adaptive routing in multicomputer networks. The following definitions develop the notation that will be used throughout the rest of the paper.

Definition 1 A *Multicomputer Network*, M, is a connected undirected graph, $M = G(N, C)$. The vertices of the graph, N, represent the set of computing nodes. The edges of the graph, C, represent the set of *bidirectional* communication channels.

Definition 2 Let $n_i \in N$ be a node of M. The set, $C_i \subseteq C$, is the set of bidirectional channels connecting n_i to its neighbors in M.

Definition 3 The *width*, W, of a channel is the number of data wires across the channel. A *flit*, or *flow control unit*, is the W parallel bits of information transferred in a single cycle. The flit is the unit used to measure the length of a packet.

Definition 4 Given a pair of nodes, n_i and n_j, the set, Q_{ij}, of *routes* joining n_i to n_j is the fixed and predetermined set of directed *acyclic* paths from the source node, n_i, to the destination node, n_j.

Definition 5 For each destination node, n_j, the *profitable* channel set, $R_{ij} \subseteq C_i$, is the subset of channels

connected to n_i, where $c_k \in R_{ij} \Rightarrow c_k \in q_m \in Q_{ij}$. In other words, forwarding a packet along the routes in Q_{ij} is equivalent to sending it out through a profitable channel in R_{ij}.

Definition 6 For each node, $n_i \in N$, the *Routing Relation* $R_i = \{(n_j, c_k) : n_j \in N - \{n_i\}, c_k \in R_{ij}\}$ defines for each possible destination node, $n_j \in N$, its corresponding profitable channel set, R_{ij}.

Definition 7 The actual path a packet traverses while in transit in the communication network is referred to as the *trajectory* of the packet. Packet trajectories are identical to the packet routes in oblivious routing schemes but are *non-deterministic* in our adaptive formulation.

We assume the following:

- Long messages are broken into packets that are the logical data entities transferred across the network.
- Packets are of fixed length; *ie*, packet length = L, where L is a network-wide constant.
- Complete routing information is included in the header flit of each packet.
- Packets are forwarded in virtual cut-through style.
- A message packet arriving at its destination node is consumed. This is commonly known as the *consumption assumption*.
- A node can generate messages destined to any other node in the network.
- Nodes can produce packets at any rate subject to the constraint of available buffer space in the network, and packets are source queued.
- Each node in the network has complete knowledge of its own routing relation.

Figure 1 presents our view of the structure of a node in a multicomputer network. Conceptually, a node can be partitioned into a computation subsystem, a communication subsystem, and a message interface. For our purpose, the computation subsystem serves as the producer and consumer of the messages routed by the communication subsystem of the node. The message interface consists of dedicated hardware that handles the overhead in sending, receiving, and reassembling message packets. Internally, the communication subsystem consists of an adaptive control and a small number of message-packet buffers. Routing decisions are made by the adaptive control, based entirely on locally available information. The bidirectional channel assumption is adopted to allow the network to exploit locality in general message-communication patterns. Furthermore, it

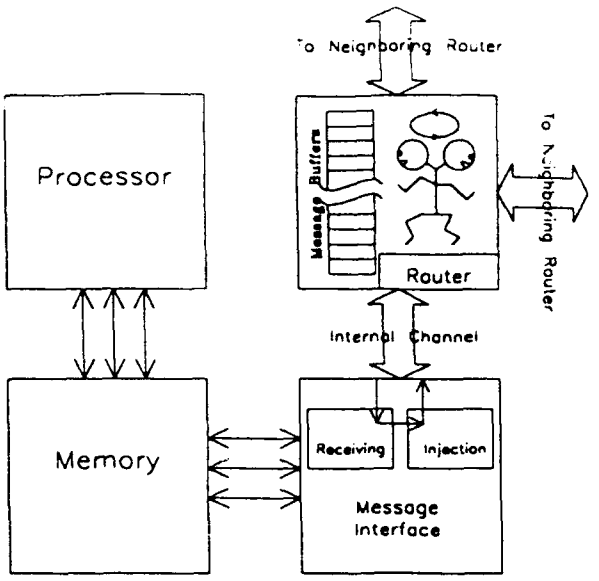


Figure 1: Structure of a node.

assures an identical number of input and output communication channels in each node, irrespective of the underlying network topology. The fixed-packet-length assumption is not essential and can be replaced by a *bounded*-packet-length assumption, *ie*, packet length $\leq L$, without invalidating any of our major results. It is adopted solely to simplify the exposition.

Communication Deadlock Freedom

In any adaptive routing scheme that allows arbitrary multipath routing, it is necessary to assure freedom from communication deadlock. Communication deadlock is caused generically by the existence of *cyclic* dependencies among communication resources along the message routes. Methods to prevent communication deadlock have been intensively researched and many schemes exist; of these, the methods of structured buffer pools [7] and virtual channels [8] are representative. In essence, all of these methods approach the problem by re-mapping any dependency that is potentially cyclic into a corresponding *acyclic* dependency structure. These methods employ restructuring techniques that require information of a global, albeit static, character. In contrast, a very simple technique that is independent of network size and topology, using voluntary *misrouting*, was suggested in [9] for networks that employ data exchange operations. Such a preemption technique utilizes only local information, and is dynamic in character. It prevents deadlock by breaking the potentially cyclic communication dependencies into disjoint paths of unit length. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly matched.

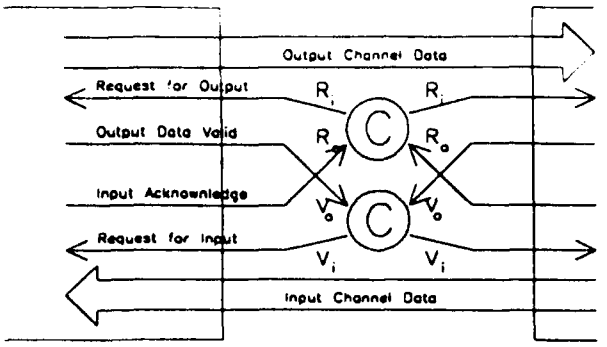


Figure 2: Two-phase protocol signaling.

A simple way is to have all bidirectional channels of the same node operate *coherently* under the protocol described next.

The Coherent Protocol. We now describe the channel data-exchange protocol in detail. It is used to match the transfer rates across all channels of the same node. The protocol employs four control signals per channel, two from each of the communicating partners, and is completely symmetric between the partners. The signaling events for a channel, $c \in C$, are:

- R_o — output event to the communicating partner indicating that this node is Ready to accept another input flit from its partner. It also serves as an acknowledgment to its partner of the successful completion of the previous transfer cycle.
- R_i^c — input event from the communicating partner indicating that the partner is Ready to accept another output flit from this node. It is also an acknowledgment from the partner of the successful completion of the previous transfer cycle.
- V_o — output event to the communicating partner indicating that the data flit values currently held at the output channel of this node are Valid and its partner should latch in the held values.
- V_i^c — input event from the communicating partner indicating that the data flit values currently asserted at the input channel of this node are Valid and the node should latch in the held values.

We proceed to define our handshaking protocol across channels of a node, $n_k \in N$, in a CSP-like notation [10]:

$$*[\quad R_o,\ [\forall c \in C_k,\ R_i^c];\quad \textit{apply out data};$$
$$\quad\ V_o,\ [\forall c \in C_k,\ V_i^c];\quad \textit{latch in data};\quad]$$

Observe that R_o and V_o denote, respectively, the *unique* outgoing Ready and data Valid signaling event to all neighbors of n_k. This enforces the matching of outgoing data rates. On the other hand, the matching of incoming data rates is enforced through the *synchronized wait* for the R_i^c and V_i^c signaling events from all neighbors. The handshaking events, R_o and R_i^c, interlock with events V_o, V_i^c to guarantee the stability and strict alternation of each other. The *initial* state of a channel has both directions of the channel ready to accept a new data flit, and proceeds thereafter in a *demand-driven* fashion. Figure 2 shows a possible conceptual realisation of the protocol under the two-phase signaling convention [11] popular for off-chip communication. Since all the handshaking events defined are local between nearest neighbors, a network following the coherent protocol is *arbitrarily extensible*.

Observe that under cut-through switching, a packet can span many different channels. An outgoing channel occupied by a packet may not be able to assert V_o until after valid data has been asserted by the corresponding incoming channel occupied by the packet; this induces matching of data rates across the two occupied channels. The notion of coherency introduced here is a natural way to accommodate such potential dependencies among the various channels of a node under cut-through switching. Another notion that arises naturally is that of a *null flit*. To effect a transfer of data in one direction of a channel while the opposite direction is idle, the receiving partner is required to transmit a null flit in order to satisfy the convention dictated by the exchange protocol.

Deadlock Freedom. We now demonstrate that to assure communication deadlock freedom for networks operating under the coherent protocol, it is sufficient to employ voluntary misrouting to prevent potential buffer overflow. To proceed, observe that routing under the cut-through switching model imposes the following integrity constraints:

1. Packets must always be forwarded to neighbors with their header flits transmitted first. In particular, voluntary misrouting of any internally buffered packet must start from the header flit of the selected packet.

2. Once the flit stream of a packet has been assigned a particular outgoing channel, the assignment must be maintained for the remaining cycles until the entire packet has been transmitted.

These constraints exist because all of the necessary routing information of a packet is encapsulated in the packet header. Interrupting a packet flit stream mid-transfer would render the latter part of the packet undeliverable. To establish deadlock freedom, it is sufficient to show that each node can *independently* complete each transfer cycle and initiate a new one, in a bounded period, without violating the stated constraints. We now

show that as long as we have an equal number of input and output channels per node, a condition satisfied readily by our bidirectional channel assumption, we can always satisfy the stated logical requirements, thereby assuring freedom from communication deadlock.

Theorem 1 Let M denote a coherent multicomputer network where each node has an equal number of input and output channels. If M employs voluntary misrouting to prevent potential buffer overflow, then it is free from deadlock.

Proof. We need to show that buffer overflow can always be prevented by misrouting without violating the cut-through switching integrity constraints. We proceed with a counting argument: Let d denote the number of channels at a node. During a protocol cycle, there may be as many as $n^* \leq d$ new data flits arriving at the input channels. A fraction of these, $0 \leq n' \leq n^*$, are new header flits; the remaining $n^* - n'$ are non-header flits of arriving packets. Of these non-header flits, a fraction of them, $0 \leq n'' \leq n^* - n'$, belong to packets that have already been assigned output channels, and the remaining $n^* - n' - n''$ flits belong to waiting packets that are buffered inside the node. Therefore, the node has at least a total of $n' + (n^* - n' - n'')$ header flits that are eligible for immediate routing. Hence, in the following cycle, a node can find at least $n' + (n^* - n' - n'') + n'' = n^*$ flits that can be transmitted or misrouted without violating the cut-through switching-integrity constraints. This assures that no buffer overflow will occur. Since the node can always complete its protocol cycles in bounded time, the network is free from deadlock. ■

Since the validity of the above proof does not depend on a node's storage capacity, deadlock freedom is established independent of the amount of available buffer space. The simple criterion of having an equal number of input and output channels is sufficient to assure deadlock freedom for a coherent network. In practice, additional buffers are needed in order to inject packets into the network, and to improve the network performance.

Network Progress Assurance

The adoption of voluntary misrouting renders communication deadlock a non-issue. However, misrouting also creates the burden of having to demonstrate progress in the form of message-delivery assurance. In particular, a network can run into a *livelock*. Consider the sequence of routing scenarios depicted in figure 3 for a bidirectional ring consisting of eight nodes and eight packets. Each of the packets consists of four data flits that span multiple channels and internal buffers. Suppose the nodes employ the following simple, deterministic, packet-to-channel assignment rule: Whenever two

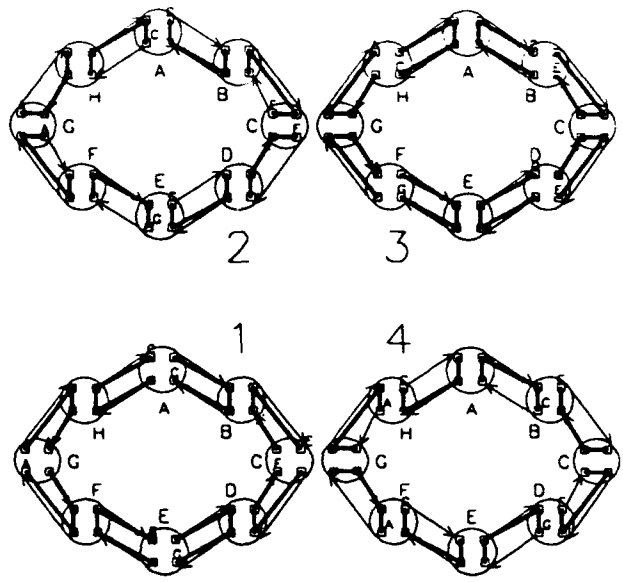


Figure 3: Livelock due to bad assignments.

incoming packets both request the same outgoing channel, the packet from the clockwise neighbor always wins. Given that, initially, nodes A, C, E, and G each receive two packets destined to nodes that are, respectively, distance two from them in the clockwise direction, after four routing cycles, the packets are all back to where they started! This example illustrates that packets can be forever denied delivery to their destinations even in the absence of communication deadlock.

Channel-access competitions are, however, not the only type of conflict that can lead to livelock. Consider the situations depicted in figure 4 for the same bidirectional ring network. The traffic patterns are coincidental in such a way that none of the packets will ever have a chance to select its own output channel; rather, at every node, each packet must be forwarded along the only remaining channel, in compliance with the voluntary misrouting discipline, in order to avoid deadlock. It is clear that no matter what assignment strategy one chooses, it is impossible to break this kind of livelock without adding extra buffers per node. In other words, additional measures and resources have to be introduced in order to assure progress, *ie*, delivery of packets, in the network.

Buffering Discipline and Requirement. In order to assure packet delivery in spite of voluntary misrouting, extra buffers are required to store packets temporarily. In particular, sufficient buffers must be provided to allow the adaptive control to give *any* newly arriving packet a chance to escape preemption if so determined by the assignment algorithm. We now demonstrate the existence of such a solution using a bounded number of

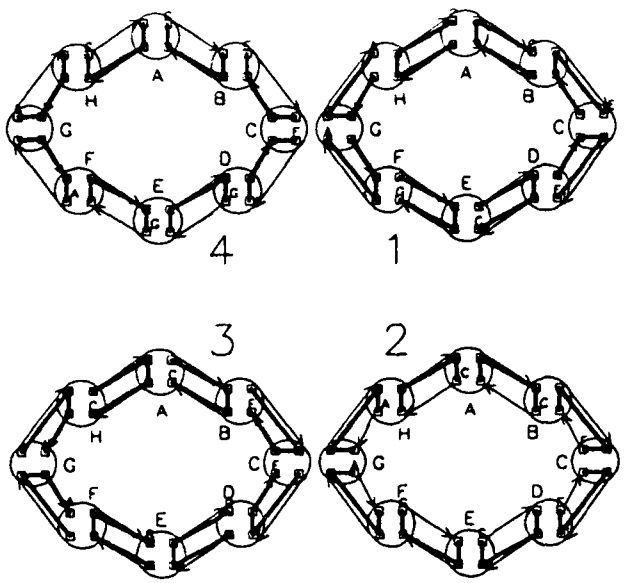


Figure 4: Livelock due to lack of assignments.

buffers. We assume the following buffering discipline:

1. Storage is divided into buffers of equal size; each is capable of holding an entire message packet.

2. Each buffer has exactly one input and one output port; this permits simultaneous reading and writing. A good example is a *FIFO* queue of length L.

3. Except as stated below, a buffer can be occupied by only one packet at a time. Oftentimes a packet may not fill its entire buffer, as in case of a partial cut-through. Such a packet occupies both the input and output ports to the buffer.

4. A buffer can be used temporarily to store two packets at a time, if and only if, one of them is leaving through the output port connected to an output channel, and the other is entering through the input port connected to an input channel.

Let b and d denote, respectively, the number of buffers and channels, *ie*, the *degree*, at each node. First, we observe that given the above buffering discipline, we must have $b \geq d$. To see this, assume that $L \gg d$, and consider the following sequence of events at a node with all buffers initially empty: At cycle $t = 0$, a packet P_0 arrives and is forwarded to its requested output channel c^* at cycle $t = 1$. Then, at cycles $t = L-d$ up to $t = L-2$, a total of $d-1$ packets, P_i, $i = 1, \ldots, d-1$, arrive one after another in $d-1$ consecutive cycles, all requesting the same output channel c^*. Finally, at cycle $t = L+2$, another packet P_d arrives, requesting the same channel c^*. The worst case happens when the

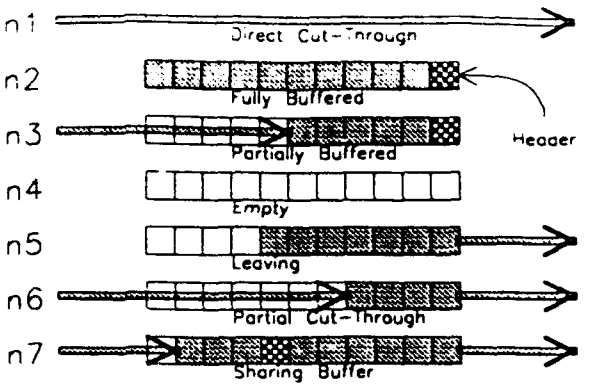


Figure 5: Accounting of buffer allocations.

assignment algorithm always favors the latest-arriving packet by requiring it to stay and avoid preemption, and has each occupy a distinct buffer. Given the above arrival sequence, at cycle $t = L+1$, packet P_{d-1} will be forwarded through c^*, which now becomes idle. As a result, each packet from P_1 up to P_d will have to be temporarily stored as it arrives. Since each packet must be allocated to a distinct buffer, we must have $b \geq d$. We now show that having $b = d$ buffers is also sufficient.

Theorem 2 Let M be a coherent network where each node has b packet buffers inside the router operating under the stated assumptions. Then $b = d$ buffers per router is necessary and sufficient to always allow at least one packet, chosen arbitrarily by the assignment algorithm at each node, to escape preemption.

Proof. Necessity follows immediately from the preceding discussion. We proceed to establish sufficiency through a counting argument. Observe that a node is required to consider misrouting of packets in the next cycle only when there are new packets arriving at the current cycle. Figure 5 depicts an accounting of all possible cases of buffer allocation at the end of any such routing cycle. Let n_1 up to n_7 denote, respectively, the number of packets or buffers in each case; and n_0 denote the number of newly arrived packets. Then, for inputs, we have $n_0+n_1+n_3+n_6+n_7 \leq d$; for outputs, we have $n_1+n_5+n_6+n_7 \leq d$. To simplify the counting argument, let us assume for the moment that $n_0 = 1$. Let P^* denote the privileged packet chosen by the assignment algorithm to stay behind and avoid misrouting in the following cycle. P^* must be either a newly arrived packet or an already buffered packet. If P^* is a buffered packet, then either the newly arriving packet finds an idle output channel to directly cut through the node, or else we must have $n_1+n_5+n_6+n_7 = d \Rightarrow n_5 \geq n_0+n_3$, which, in turn, implies that there will always be an available buffer ready to accept it. On the other hand, if P^* is a newly arriv-

ing packet, then either $n_4 + n_5 > 0$, and, hence, there is a buffer ready to accept it, or else we must have $n_2 + n_3 + n_6 + n_7 = b = d$. This, together with the above inequality on inputs, $\Rightarrow n_2 \geq n_0 + n_1 \Rightarrow n_2 > 0$. Furthermore, $n_0 > 0 \Rightarrow n_1 + n_6 + n_7 < d$. In other words, the packet will be able to find at least one buffer with a full idle packet as well as an idle output channel to preempt this idle packet and thus make room for itself. This establishes the validity for single-packet arrivals. Finally, repeated applications of the above argument then establish the validity for multiple packet arrivals, and, therefore, the sufficiency condition. ■

The trick in allowing the escape from misrouting for any arbitrarily chosen packet is to provide at least a critical, minimum number of buffers that is sufficient to assure either that empty buffers still exist, or that all buffers have been occupied, and, hence, that there is some other packet that can be misrouted instead. The particular number required depends on the adopted buffering structure and discipline; adding more buffers per node will allow the assignment algorithm to operate with more flexibility and perform better. In any case, by having a sufficient number of buffers, competition of profitable channel access is transformed into a competition for the right to stay behind and wait until the winner's profitable channel becomes available, at which time it will be forwarded. Hence, winners chosen by the assignment algorithm will have the chance to follow the actual paths determined by the routing relations. In a sense, assurance of packet delivery has now been reduced to that of picking *consistent* winners across the network.

Packet-Priority Assignments. An effective scheme for picking consistent winners that is independent of any particular network topology is to resolve the channel-access conflicts according to a *priority* assignment. In particular, the process of forwarding a packet towards its destination can be viewed as a sequence of actions performed to reduce the packet's distance from destination, provided that the set $\mathcal{R} = \{R_i\}$ of routing relations is defined in terms of an underlying metric of the network. In this case, as the result of a channel-access conflict, the winner will be routed along a profitable channel, thus decreasing its distance from the destination. The losers, depending on whether or not they are misrouted along the remaining unprofitable channels, may or may not increase their distance from destination. Ideally, one would prefer a strict monotonic decrease of distance to destination for each packet routed in the network. As this is impossible under our adaptive model, the alternative is to ensure monotonic decrease over a sequence of exchanges involving multiple packets. This can be achieved by giving higher priority to packets with shorter distances from destination then to those with longer distances, as follows:

$$P_1 > P_2 \Longleftrightarrow D_1 < D_2,$$

where P is a packet's priority and D its distance from destination. We now show that this is sufficient to guarantee livelock freedom.

Theorem 3 A packet-to-channel assignment strategy that observes the defined distance priority, together with the set $\mathcal{R}$ of metric-based routing relations, guarantees livelock freedom in a network.

Proof. At the beginning of a routing cycle, let $D > 0$ be the minimum packet distance from destination. During this cycle, a packet with distance D competes with other packets for channels leading to its destination. If it wins the competition, it will be forwarded along a profitable channel within L cycles. It it loses, it must be to another packet also distance D away from its destination, according to the defined priority. In both cases, the minimum distance is reduced to $< D$ within L cycles. Therefore, D will eventually be reduced to zero, in which case a successful packet delivery occurs and the above argument can be applied again to assure repeated deliveries. This establishes livelock freedom. ■

Observe that although the distance priority alone suffices to guarantee global progress in a message network, no corresponding statement can be made concerning each individual packet. This is because it is possible for packets that are far away from their destinations to be repeatedly defeated by newly injected packets that are closer to their respective destinations. A more complex priority scheme that assures delivery of *every* packet can be obtained by augmenting the above simple scheme with *age* information, with higher priorities assigned to older packets:

$$(A_1, D_1) > (A_2, D_2) \Longleftrightarrow (A_1 > A_2) \vee ((A_1 = A_2) \wedge (D_1 < D_2)),$$

where A is a packet's *age*, that is, the number of routing cycles elapsed since the injection of the packet. Empirical simulation results indicate that the simple distance-assignment scheme is sufficient for almost all situations, except under an extremely heavy applied load.

Network-Access Assurance

A different kind of progress assurance that requires demonstration under our adaptive formulation is the ability of a node to inject packets eventually. Because of the requirement to maintain strict balance of input and output data rates, a node located in the center of heavy traffic might be denied access to the network indefinitely. Figure 6 depicts a possible conceptual realization of a message interface. Its operation is similar to the register insertion ring interface described in

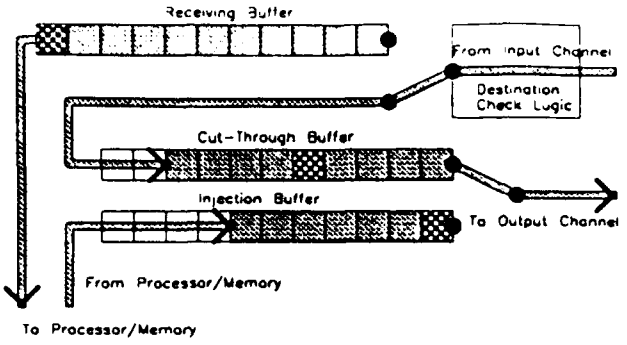


Figure 6: Inside the message interface.

[12]. It uses two FIFO buffers that can be connected to the output channel towards the network via a switch. Whenever the node has a packet to transmit, it loads the packet into the injection buffer as soon as the buffer becomes empty. When message traffic arrives from the network input channel, it passes through the destination check logic, which redirects any traffic destined to this node to the node memory. Any remaining passing traffic is loaded into the cut-through buffer, which is normally connected to the output channel. Whenever the cut-through buffer becomes empty, the control logic checks to see if there is an output packet waiting for injection. In such case, the switch is toggled so that the output channel is connected to the injection buffer, and the injection proceeds. As the output packet is being forwarded, any passing traffic is loaded into the cut-through buffer. The switch connection is flipped back to the cut-through buffer after injection has been finished, and the process repeats. The main interesting property of the message interface for our current discussion is that it provides the mechanism to capture and accumulate interpacket gaps, which need not be contiguous, as empty spaces inside the cut-through buffers. When enough space has been collected, *ie*, the entire packet length, hence, an entire empty buffer, another new packet can be injected into the network. With such a mechanism, the question of assuring eventual packet injection is translated into that of assuring arrival of enough interpacket gaps whenever a node has a packet injection outstanding.

Round-Trip Packets. One simple way to assure network access is to have each packet delivered by the network be returned to its original sender upon arrival at its destination. Since each message interface starts with an empty injection buffer, consumption of its own round-trip packets will always restore its ability to inject the next source-queued packet. More sophisticated versions of such a scheme will use several cut-through buffers, and will demand that packets be returned only if the stock of empty cut-through buffers has been depleted below a predetermined threshold. In this way, the number of round-trip packets can be dramatically reduced when traffic is relatively moderate. Unfortunately, as traffic density increases, the population of round-trip packets also increases, thus further decreasing useful network bandwidth.

Packet-Injection Control. A different scheme that does not incur this overhead is to have the nodes maintain a bounded synchrony with neighbors on the total number of injections. Nodes that fall behind will, in effect, prohibit others from injecting until they catch up. We shall adopt the convention that a node having no packet to inject has a *null* packet queued up; *ie*, during each routing cycle, every node either has a null or real packet ready to inject or else is in the process of injecting a real packet. The null-packet convention is required to prevent quiescent nodes that do not have any packet to inject from blocking injections in the active nodes. Our scheme is to introduce *local synchronization* among neighboring nodes such that the total number of packets injected by a node after each routing cycle will not differ by more than K, a positive constant, from those of its neighbors. We assume that each node explicitly maintains records of the total number of packet injections made by each of its neighbors, measured *relative to that of its own,* and that the information required to update these records in each node is exchanged on separate direct links between the message interfaces among neighbors. A node is allowed to inject its queued packet only if its own number of total injections is fewer than K packet injections ahead of its minimum neighbor. Nodes that are allowed to inject will examine their queued packets. Null packets are always injected by convention, whereas real packets are injected only if the injection mechanism described previously finds at least one empty buffer available to absorb the injection transient. We now show that, with eventual delivery of the packets already injected, this injection synchronisation protocol establishes cooperation among the nodes to assure the eventual occurrence of empty cut-through buffers in the message interface for nodes that have real packets waiting for injection as permitted by the protocol.

Lemma 4 A node that has a packet waiting for injection that is permissible under the above injection protocol will eventually inject.

Proof. Observe that, by convention, if the pending packet is null, the node is able to inject immediately, so that the lemma is true vacuously. We now proceed to establish its validity for real packets. Suppose, to the contrary, that a particular node, $n \in N$, is blocked from injection indefinitely because the injection mechanism cannot accumulate sufficient empty buffer space

to absorb the injection transient. Our injection protocol then dictates that its neighbors also will be blocked indefinitely from injecting. These, in turn, indefinitely block their neighbors, and so on. Given a finite network, all nodes are eventually blocked from any further injection, and eventually *no* new packet can enter the network. Given the eventual delivery guarantee for packets already injected, ultimately the network will be void of packets; at that point, the input channel to the interface of n will become idle, thus enabling it to resume the accumulation of empty spaces inside the cut-through buffer. Eventually, it will have collected enough spaces to enable the injection of its queued packet into the network. This contradicts the original indefinite blocking assumption of n, and thereby establishes the validity of the lemma. ■

We are now ready to show that by following the above injection protocol every individual node will eventually be permitted to inject, and, hence, according to the above lemma, all will eventually inject. Specifically, let M be a network, and let T_i denote the total number of packet injections from node $n_i \in N$ since initialization. We now prove that T_i is strictly increasing over time.

Theorem 5 Given the injection protocol and a finite network that is livelock free, the total number of packet injections for each node strictly increases over time.

Proof. During a routing cycle, let $t = \min_{n_i \in N} T_i$ denote the minimum among numbers of packet injections since initialization, taken over all the nodes of the network, and let $S = \{n_i \in N | T_i = t\}$ denote the set of nodes that have recorded the minimum number of packet injections since initialization. Since $K > 0$, according to our protocol, every node $n \in S$ is permitted to inject. Lemma 4 then guarantees eventual injections from all of the nodes in S; hence, t, the minimum number of packet injections per node, is guaranteed to eventually increase over time. This, in turn, guarantees that T_i strictly increases over time, $\forall n_i \in N$. ■

Hence, we are assured of eventual packet injection for each individual node of the network. In other words, the above theorem establishes *fairness* in network access among all the nodes.

Performance Comparisons

An extensive set of simulations was conducted to obtain information concerning the potential gain in performance by switching from the oblivious wormhole to the adaptive cut-through technique. We now summarize very briefly the typical kind of behaviors observed in these simulations. A much more detailed discussion can be found in [5]. Among the various statistics collected, the two most important performance metrics in communication networks are *network throughput* and *message latency*. Figure 7 plots the sustained normalized network throughput versus the normalised applied load of the oblivious and adaptive schemes for a 16×16 2D-mesh network under random traffic. The normalisation is performed with respect to the network bisection bandwidth limit. Starting at a very low applied load, the throughput curves of both schemes rise along a unit slope line. The oblivious wormhole curve levels off at $\approx 45 - 50\%$ of normalised throughput but remains *stable* even under an increasingly heavy applied load. In contrast, the adaptive cut-through curve keeps rising along the unit slope line until it is out of the range of collected data. It should be pointed out, however, that the increase in throughput obtained is also partly due to the extra silicon area invested in buffer storage, which makes adaptive choices available.

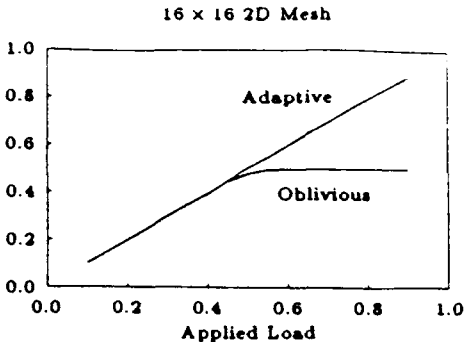


Figure 7: Throughput versus applied load.

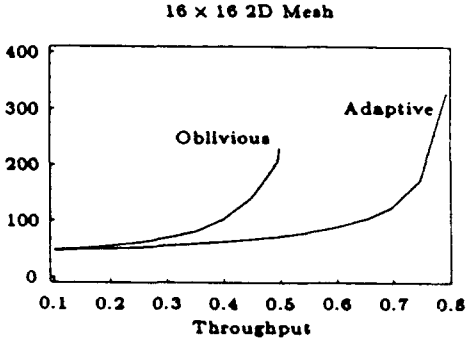


Figure 8: Message latency versus throughput.

Figure 8 plots the message latency versus normalized throughput for the same 2D-mesh network for a typical message length of 32 flits. The curves shown are typical of latency curves obtained in virtual cut-through switching [6]. Both curves start with latency values close to the ideal at very low throughput, and remain relatively flat until they hit their respective transition points, after which both rise rapidly. The transition points are $\approx 40\%$ and 70%, respectively, for the oblivi-

ous and adaptive schemes. In essence, adaptive routing control increases the quantity of routing service, *ie*, network throughput, without sacrificing the quality of the provided service, *ie*, message latency, at the expense of requiring more silicon area.

Summary

Several issues related to adaptive cut-through routing have been addressed in the course of this research, and we did not encounter any insurmountable problem. Rather, the simplicity of these resolution mechanisms gives us hope that the adaptive scheme can be made to improve on the already highly evolved oblivious routing scheme. The discussion in this paper has focused on issues concerning the *feasibility* of the proposed adaptive routing framework. Within this framework, we also have studied and found promising approaches to fault-tolerant routing. Clearly, more work remains to be done. Perhaps the most challenging of all is to realize on *silicon* the set of ideas outlined in this study.

References

[1] Charles L. Seitz, "The Cosmic Cube," *CACM*, 28(1): 22–33, January 1985.

[2] William C. Athas and Charles L. Seitz., "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer*: 9–24, August 1988.

[3] William J. Dally and Charles L. Seitz, "The Torus Routing Chip," *Distributed Computing* (1): 187–196, 1986.

[4] Charles M. Flaig, *VLSI Mesh Routing Systems*. Caltech Computer Science Department Technical Report, 5241:TR:87.

[5] John Y. Ngai, *Adaptive Routing in Multicomputer Networks*. Ph.D. Thesis, Computer Science Department, Caltech. To be published.

[6] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks* 3(4): 267–286, Sept. 1979.

[7] P. Merlin, and P. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks — I : Store-and Forward Deadlock," *IEEE Transactions on Communications*, Vol. COM-28(3): 345–354, March 1980.

[8] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36(5): 547–553, May 1987.

[9] A. Borodin, and J. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation," *Journal of Computer and System Sciences* 30: pp. 130–145, 1985.

[10] Alain J. Martin, "A Synthesis Method for Self-timed VLSI Circuits," *Proc. 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, IEEE Comp. Soc. Press: 224–229, 1987.

[11] Charles L. Seitz, "System Timing," *Introduction to VLSI Systems*, Chapter 7, C. Mead & L. Conway, Addison-Wesley, 1980.

[12] M. T. Liu, "Distributed Loop Computer Networks," *Advances in Computers*, M. Yovits, Academic Press: 163–221, 1978.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports
18 October 1989
Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check for a minimum of $50.00 drawn on a U.S. bank in U.S. currency, payable to CALTECH.

___CS-TR-89-11 $9.00 *Reactive-Process Programming and Distributed Discrete-Event Simulation*, PhD Thesis
Su, Wen-King

___CS-TR-89-10 $7.00 *Silicon Models of Early Audition*, PhD Thesis
Lazarro, John

___CS-TR-89-09 $15.00 *A Framework for Adaptive Routing in Multicomputer Networks*, PhD Thesis
Ngai, John

___CS-TR-89-07 $6.00 *Constraint Methods for Neural Networks and Computer Graphics*, PhD Thesis
Platt, John

___CS-TR-89-06 $1.00 *The First Asynchronous Microprocessor: The Test Results*
Martin, Alain J, Steven M Burns, T K Lee, Drazen Borkovic, and Pieter J Hazewindus

___CS-TR-89-05 $2.00 *The Essence of Distributed Snapshots*
Chandy, K. Mani

___CS-TR-89-04 $5.00 *Submicron Systems Architecture Project*
ARPA Semiannual Technical Report

___CS-TR-89-03 $3.00 *Feature-oriented Image Enhancement with Shock Filters, I*
Rudin, Leonid I with Stanley Osher

___CS-TR-89-02 $3.00 *Design of an Asynchronous Microprocessor*
Martin, Alain J

___CS-TR-89-01 $4.00 *Programming in VLSI From Communicating Processes to Delay-insensitive Circuits*,
Martin, Alain J

___CS-TR-88-22 $2.00 *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm*,
Su, Wen-King and Charles L Seitz

___CS-TR-88-21 $3.00 *Winner-Take-All Networks of O(N) Complexity*,
Lazzaro, John, with S Ryckebusch, M A Mahowald and C A Mead

___CS-TR-88-20 $7.00 *Neural Network Design and the Complexity of Learning*,
Judd, J Stephen

___CS-TR-88-19 $5.00 *Controlling Rigid Bodies with Dynamic Constraints*,
Barzel, Ronen

___CS-TR-88-18 $3.00 *Submicron Systems Architecture Project*,
ARPA Semiannual Technical Report

___CS-TR-88-17 $3.00 *Constrained Differential Optimization for Neural Networks*,
Platt, John C and Alan H Barr

___CS-TR-88-16 $3.00 *Programming Parallel Computers*,
Chandy, K Mani

___CS-TR-88-15 $13.00 *Applications of Surface Networks to Sampling Problems in Computer Graphics*, PhD Thesis
Von Herzen, Brian

___CS-TR-88-14 $2.00 *Syntax-directed Translation of Concurrent Programs into Self-timed Circuits*
Burns, Steven M and Alain J Martin

___CS-TR-88-13 $2.00 *A Message-Passing Model for Highly Concurrent Computation*,
Martin, Alain J

Caltech Computer Science Technical Reports

___CS-TR-88-12 $4.00 *A Comparison of Strict and Non-strict Semantics for Lists*, MS Thesis
Burch, Jerry R

___CS-TR-88-11 $5.00 *A Study of Fine-Grain Programming Using Cantor*, MS Thesis
Boden, Nanette J

___CS-TR-88-10 $3.00 *The Reactive Kernel*, MS Thesis
Seizovic, Jacov

___CS-TR-88-07 $3.00 *The Hexagonal Resistive Network and the Circular Approximation*,
Feinstein, David I

___CS-TR-88-06 $3.00 *Theorems on Computations of Distributed Systems*,
Chandy, K Mani

___CS-TR-88-05 $3.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

___CS-TR-88-04 $3.00 *Cochlear Hydrodynamics Demystified*
Lyon, Richard F and Carver A Mead

___CS-TR-88-03 $4.00 *PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics*,
MS Thesis
Gupta, Rajiv

___CS-TR-88-02 $4.00 *Automated Compilation of Concurrent Programs into Self-timed Circuits*, MS Thesis
Burns, Stephen M

___CS-TR-88-01 $3.00 *C Programmer's Abbreviated Guide to Multicomputer Programming*,
Seitz, Charles, Jakov Seizovic and Wen-King Su

___5258:TR:88 $3.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

___5256:TR:87 $2.00 *Synthesis Method for Self-timed VLSI Circuits*,
Martin, Alain
current supply only: see *Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design*, 224-229, Oct'87

___5253:TR:88 $2.00 *Synthesis of Self-Timed Circuits by Program Transformation*,
Burns, Steven M and Alain J Martin

___5251:TR:87 $2.00 *Conditional Knowledge as a Basis for Distributed Simulation*,
Chandy, K. Mani and Jay Misra

___5250:TR:87 $10.00 *Images, Numerical Analysis of Singularities and Shock Filters*, PhD Thesis
Rudin, Leonid Iakov

___5249:TR:87 $6.00 *Logic from Programming Language Semantics*, PhD Thesis
Choo, Young-il

___5247:TR:87 $6.00 *VLSI Concurrent Computation for Music Synthesis*, PhD Thesis
Wawrzynek, John

___5246:TR:87 $3.00 *Framework for Adaptive Routing*
Ngai, John Y and Charles L. Seitz

___5244:TR:87 $3.00 *Multicomputers*
Athas, William C and Charles L Seitz

___5243:TR:87 $5.00 *Resource-Bounded Category and Measure in Exponential Complexity Classes*, PhD Thesis
Lutz, Jack H

___5242:TR:87 $8.00 *Fine Grain Concurrent Computations*, PhD Thesis
Athas, William C.

___5241:TR:87 $3.00 *VLSI Mesh Routing Systems*, MS Thesis
Flaig, Charles M

___5240:TR:87 $2.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

___5239:TR:87 $3.00 *Trace Theory and Systolic Computations*
Rem, Martin

___5238:TR:87 $7.00 *Incorporating Time in the New World of Computing System*, MS Thesis
Poh, Hean Lee

5236:TR:86 $4.00 *Approach to Concurrent Semantics Using Complete Traces*, MS Thesis
Van Horn, Kevin S.

5235:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

5234:TR:86 $3.00 *High Performance Implementation of Prolog*
Newton, Michael O

5233:TR:86 $3.00 *Some Results on Kolmogorov-Chaitin Complexity*, MS Thesis
Schweizer, David Lawrence

5232:TR:86 $4.00 *Cantor User Report*
Athas, W.C. and C. L. Seitz

5230:TR:86 $24.00 *Monte Carlo Methods for 2-D Compaction*, PhD Thesis
Mosteller, R.C.

5229:TR:86 $4.00 *anaLOG - A Functional Simulator for VLSI Neural Systems*, MS Thesis
Lazzaro, John

5228:TR:86 $3.00 *On Performance of k-ary n-cube Interconection Networks*,
Dally, Wm. J

5227:TR:86 $18.00 *Parallel Execution Model for Logic Programming*, PhD Thesis
Li, Pey-yun Peggy

5223:TR:86 $15.00 *Integrated Optical Motion Detection*, PhD Thesis
Tanner, John E.

5221:TR:86 $3.00 *Sync Model: A Parallel Execution Method for Logic Programming*
Li, Pey-yun Peggy and Alain J. Martin
current supply only: see *Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86*

5220:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

5215:TR:86 $2.00 *How to Get a Large Natural Language System into a Personal Computer*,
Thompson, Bozena H. and Frederick B. Thompson

5214:TR:86 $2.00 *ASK is Transportable in Half a Dozen Ways*,
Thompson, Bozena H. and Frederick B. Thompson

5212:TR:86 $2.00 *On Seitz' Arbiter*,
Martin, Alain J

5210:TR:86 $2.00 *Compiling Communicating Processes into Delay-Insensitive VLSI Circuits*,
Martin, Alain
current supply only: see *Distributed Computing* v 1 no 4 (1986)

5207:TR:86 $2.00 *Complete and Infinite Traces: A Descriptive Model of Computing Agents*,
van Horn, Kevin

5205:TR:85 $2.00 *Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity*,
Schweizer, David and Yaser Abu-Mostafa

5204:TR:85 $3.00 *An Inverse Limit Construction of a Domain of Infinite Lists*,
Choo, Young-Il

5202:TR:85 $15.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5200:TR:85 $18.00 *ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation*, PhD thesis
Whelan, Dan

5198:TR:85 $8.00 *Neural Networks, Pattern Recognition and Fingerprint Hallucination*, PhD thesis
Mjolsness, Eric

5197:TR:85 $7.00 *Sequential Threshold Circuits*, MS thesis
Platt, John

5195:TR:85 $3.00 *New Generalization of Dekker's Algorithm for Mutual Exclusion*,
Martin, Alain J
current supply only: see *Information Processing Letters*, **23**, 295-297 (1986)

5194:TR:85 $5.00 *Sneptree - A Versatile Interconnection Network*,
Li, Pey-yun Peggy and Alain J Martin

__5193:TR:85 $2.00 *Delay-insensitive Fair Arbiter*
Martin, Alain J

__5190:TR:85 $3.00 *Concurrency Algebra and Petri Nets*,
Choo, Young-il

__5189:TR:85 $10.00 *Hierarchical Composition of VLSI Circuits*, PhD Thesis
Whitney, Telle

__5185:TR:85 $11.00 *Combining Computation with Geometry*, PhD Thesis
Lien, Sheue-Ling

__5184:TR:85 $7.00 *Placement of Communicating Processes on Multiprocessor Networks*, MS Thesis
Steele, Craig

__5179:TR:85 $3.00 *Sampling Deformed, Intersecting Surfaces with Quadtrees*, MS Thesis,
Von Herzen, Brian P.

__5178:TR:85 $9.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

__5174:TR:85 $7.00 *Balanced Cube: A Concurrent Data Structure*,
Dally, William J and Charles L Seitz

__5172:TR:85 $6.00 *Combined Logical and Functional Programming Language*,
Newton, Michael

__5168:TR:84 $3.00 *Object Oriented Architecture*,
Dally, Bill and Jim Kajiya

__5165:TR:84 $4.00 *Customizing One's Own Interface Using English as Primary Language*,
Thompson, B H and Frederick B Thompson

__5164:TR:84 $13.00 *ASK French - A French Natural Language Syntax*, MS Thesis
Sanouillet, Remy

__5160:TR:84 $7.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

__5158:TR:84 $6.00 *VLSI Architecture for Sound Synthesis*,
Wawrzynek, John and Carver Mead

__5157:TR:84 $15.00 *Bit-Serial Reed-Solomon Decoders in VLSI*, PhD Thesis
Whiting, Douglas

__5147:TR:84 $4.00 *Networks of Machines for Distributed Recursive Computations*,
Martin, Alain and Jan van de Snepscheut

__5143:TR:84 $5.00 *General Interconnect Problem*, MS Thesis
Ngai, John

__5140:TR:84 $5.00 *Hierarchy of Graph Isomorphism Testing*, MS Thesis
Chen, Wen-Chi

__5139:TR:84 $4.00 *HEX: A Hierarchical Circuit Extractor*, MS Thesis
Oyang, Yen-Jen

__5137:TR:84 $7.00 *Dialogue Designing Dialogue System*, PhD Thesis
Ho, Tai-Ping

__5136:TR:84 $5.00 *Heterogeneous Data Base Access*, PhD Thesis
Papachristidis, Alex

__5135:TR:84 $7.00 *Toward Concurrent Arithmetic*, MS Thesis
Chiang, Chao-Lin

__5134:TR:84 $2.00 *Using Logic Programming for Compiling APL*, MS Thesis
Derby, Howard

__5133:TR:84 $13.00 *Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems*, PhD Thesis
Lin, Tzu-mu

__5132:TR:84 $10.00 *Switch Level Fault Simulation of MOS Digital Circuits*, MS Thesis
Schuster, Mike

__5129:TR:84 $5.00 *Design of the MOSAIC Processor*, MS Thesis
Lutz, Chris

5128:TM:84 $3.00 *Linguistic Analysis of Natural Language Communication with Computers*,
Thompson, Bozena H

5125:TR:84 $6.00 *Supermesh*, MS Thesis
Su, Wen-king

5123:TR:84 $14.00 *Mossim Simulation Engine Architecture and Design*,
Dally, Bill

5122:TR:84 $8.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5114:TM:84 $3.00 *ASK As Window to the World*,
Thompson, Bozena, and Fred Thompson

5112:TR:83 $22.00 *Parallel Machines for Computer Graphics*, PhD Thesis
Ulner, Michael

5106:TM:83 $1.00 *Ray Tracing Parametric Patches*,
Kajiya, James T

5104:TR:83 $9.00 *Graph Model and the Embedding of MOS Circuits*, MS Thesis
Ng, Tak-Kwong

5094:TR:83 $2.00 *Stochastic Estimation of Channel Routing Track Demand*,
Ngai, John

5092:TM:83 $2.00 *Residue Arithmetic and VLSI*,
Chiang, Chao-Lin and Lennart Johnsson

5091:TR:83 $2.00 *Race Detection in MOS Circuits by Ternary Simulation*,
Bryant, Randal E

5090:TR:83 $9.00 *Space-Time Algorithms: Semantics and Methodology*, PhD Thesis
Chen, Marina Chien-mei

5089:TR:83 $10.00 *Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits*,
Lin, Tzu-Mu and Carver A Mead

5086:TR:83 $4.00 *VLSI Combinator Reduction Engine*, MS Thesis
Athas, William C Jr

5082:TR:83 $10.00 *Hardware Support for Advanced Data Management Systems*, PhD Thesis
Neches, Philip

5081:TR:83 $4.00 *RTsim - A Register Transfer Simulator*, MS Thesis
Lam, Jimmy
current supply only: see *Acta Informatica* 20, 301-313, (1983)

5074:TR:83 $10.00 *Robust Sentence Analysis and Habitability*,
Trawick, David

5073:TR:83 $12.00 *Automated Performance Optimization of Custom Integrated Circuits*, PhD Thesis
Trimberger, Steve

5065:TR:82 $3.00 *Switch Level Model and Simulator for MOS Digital Systems*,
Bryant, Randal E

5054:TM:82 $3.00 *Introducing ASK, A Simple Knowledgeable System*, Conf on App'l Natural Language Processing
Thompson, Bozena H and Frederick B Thompson

5051:TM:82 $2.00 *Knowledgeable Contexts for User Interaction*, Proc Nat'l Computer Conference
Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho

5035:TR:82 $9.00 *Type Inference in a Declarationless, Object-Oriented Language*, MS Thesis
Holstege, Eric

5034:TR:82 $12.00 *Hybrid Processing*, PhD Thesis
Carroll, Chris

5033:TR:82 $4.00 *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*,
Schuster, Mike, Randal Bryant and Doug Whiting

5029:TM:82 $4.00 *POOH User's Manual*,
Whitney, Telle

Caltech Computer Science Technical Reports

____5018:TM:82 $2.00 *Filtering High Quality Text for Display on Raster Scan Devices*,
Kajiya, Jim and Mike Ullner

____5017:TM:82 $2.00 *Ray Tracing Parametric Patches*,
Kajiya, Jim

____5015:TR:83 $15.00 *VLSI Computational Structures Applied to Fingerprint Image Analysis*,
Megdal, Barry

____5014:TR:82 $15.00 *Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, PhD Thesis
Lang, Charles R Jr

____5012:TM:82 $2.00 *Switch-Level Modeling of MOS Digital Circuits*,
Bryant, Randal

____5000:TR:82 $6.00 *Self-Timed Chip Set for Multiprocessor Communication*, MS Thesis
Whiting, Douglas

____4684:TR:82 $3.00 *Characterization of Deadlock Free Resource Contentions*,
Chen, Marina, Martin Rem, and Ronald Graham

____4655:TR:81 $20.00 *Proc Second Caltech Conf on VLSI*,
Seitz, Charles, ed.

____3760:TR:80 $10.00 *Tree Machine: A Highly Concurrent Computing Environment*, PhD Thesis
Browning, Sally

____3759:TR:80 $10.00 *Homogeneous Machine*, PhD Thesis
Locanthi, Bart

____3710:TR:80 $10.00 *Understanding Hierarchical Design*, PhD Thesis
Rowson, James

____3340:TR:79 $26.00 *Proc. Caltech Conference on VLSI (1979)*,
Seitz, Charles, ed

____2276:TM:78 $12.00 *Language Processor and a Sample Language*,
Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

Name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

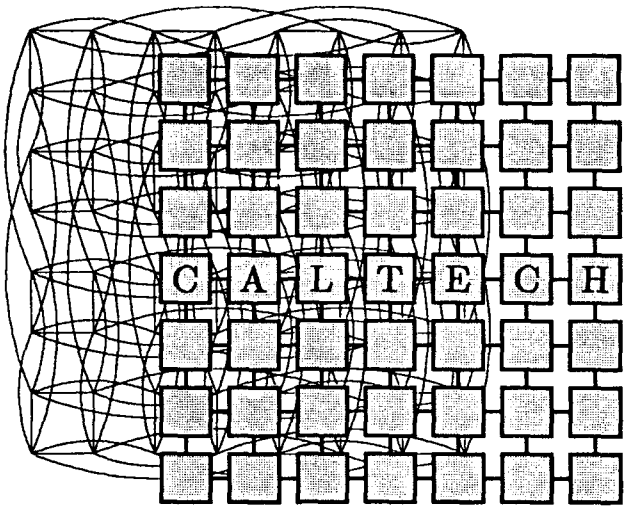
_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

_____89-11	_____88-03	_____5223	_____5164	_____5089
_____89-10	_____88-02	_____5221	_____5160	_____5086
_____89-09	_____88-01	_____5220	_____5158	_____5082
_____89-07	_____5258	_____5215	_____5157	_____5081
_____89-06	_____5256	_____5214	_____5147	_____5074
_____89-05	_____5253	_____5212	_____5143	_____5073
_____89-04	_____5251	_____5210	_____5140	_____5065
_____89-03	_____5250	_____5207	_____5139	_____5054
_____89-02	_____5249	_____5205	_____5137	_____5051
_____89-01	_____5247	_____5204	_____5136	_____5035
_____88-22	_____5246	_____5202	_____5135	_____5034
_____88-21	_____5244	_____5200	_____5134	_____5033
_____88-20	_____5243	_____5198	_____5133	_____5029
_____88-19	_____5242	_____5197	_____5132	_____5018
_____88-18	_____5241	_____5195	_____5129	_____5017
_____88-17	_____5240	_____5194	_____5128	_____5015
_____88-16	_____5239	_____5193	_____5125	_____5014
_____88-15	_____5238	_____5190	_____5123	_____5012
_____88-14	_____5236	_____5189	_____5122	_____5000
_____88-13	_____5235	_____5185	_____5114	_____4684
_____88-12	_____5234	_____5184	_____5112	_____4655
_____88-11	_____5233	_____5179	_____5106	_____3760
_____88-10	_____5232	_____5178	_____5104	_____3759
_____88-07	_____5230	_____5174	_____5094	_____3710
_____88-06	_____5229	_____5172	_____5092	_____3340
_____88-05	_____5228	_____5168	_____5091	_____2276
_____88-04	_____5227	_____5165	_____5090	_____



SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-89-4

31 March 1989

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-89-4

31 March 1989

Reporting Period:	1 November 1988 – 31 March 1989
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	K. Mani Chandy
	Alain J. Martin
	Charles L. Seitz
	Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of research activities and results for the five-month period, 1 November 1988 to 31 March 1989, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- Mosaic prototype approaching completion (2.1).
- Delivery of 2nd-generation multicomputers (2.2)
- Programming with composition (3.3)
- First asynchronous microprocessor (4.1).
- Fast self-timed mesh routing chips (4.2).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jordan Holt, Jakov Seizovic, Don Speck, Wen-King Su, Steve Taylor, Tony Wittry

The Mosaic C is an experimental fine-grain multicomputer, currently in development. Each Mosaic node is a single VLSI chip containing a 16-bit processor, a three-dimensional mesh router with each of its channels operating at 160Mb/s, a packet interface, at least 8KB of RAM, and a ROM that holds self-test and bootstrap code. These nodes are arrayed logically and physically in a three-dimensional mesh. We are working toward building a 16K-node (32×32×16) Mosaic prototype, together with the system software and programming tools required to develop application programs.

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small. The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system.

The Mosaic project includes many subtasks, which are listed below together with their current status:

Design, layout, and verification of the single-chip Mosaic node. The design and layout of the Mosaic C chip are now complete, and are going through extensive switch-level simulation tests, including the simulation of multiple nodes (see section 4.3). We expect to send a memoryless version of the node element to fabrication in about two weeks as a final check of the processor, packet interface, and router sections. These chips will be connected to external RAM and ROM to provide functional node elements for software development and host interfaces. Fabrication of the first chips in 1.2μm CMOS technology with RAM and ROM is anticipated in June 1989; quantity fabrication is anticipated in September 1989.

Internal self-test and bootstrap code. Since the Mosaic C is a programmable computing element, devoting a portion of the bootstrap ROM to self-testing greatly simplifies the logistics of producing these chips in significant quantity. The bootstrap and self-test code has been designed and is currently being written. The code will be tested using the ROM connected to the memoryless Mosaic C elements. Additional tests to the channels, which must be accomplished by the fabricator's automatic test equipment, are also being written.

Packaging. A preliminary packaging design based on TAB-packaged Mosaic

C chips was completed following a visit to Hewlett-Packard NID to understand their TAB packaging capabilities. The manufacturing and replacement unit contains eight nodes in a logical 2×2×2 submesh on a circuit-card module whose physical dimensions are approximately 2.5×5inches^2. These modules have stacking connectors that provide 160 pins on both the top and bottom, and are confined by pressure between motherboards to provide a three-dimensional connection structure that can be disassembled and reassembled for repair.

Cantor runtime system. A complete Cantor runtime system was written in Mosaic assembly code, and is now running correctly with a suite of small test programs under a Mosaic simulator on our medium-grain multicomputers (see section 3.1). This system provides the low-level implementation of message and process-creation primitives, and normally will be loaded as part of the Mosaic system initialization. The evolution of the Cantor programming language and the experience gained by use are two factors that are expected to affect continuing refinements to this system.

Cantor language, compiler, and application studies. A definition of a version of Cantor (3.0) with functions and limited message discretion was proposed in January 1989 by William C. Athas of UT Austin. We have been studying the changes in the runtime support that will be required by these improvements. In the interim, the definition and compiler implementation of Cantor 2.2 remain in use for application development.

Host interfaces and displays. The three-dimensional mesh structure of the Mosaic allows a very large bandwidth around the mesh edges. In order to initiate and interact with computations within the Mosaic, we must provide interfaces between the Mosaic message network and conventional computers and networks. One approach being studied is to use a memoryless Mosaic with a two-ported external memory as a convenient interface to workstation computers. Another external connection that is desired is a display interface. An elegant method that uses one 32×32 plane of a Mosaic as a rendering engine, frame buffer, and output video-conversion system has been developed. The detailed design of the video output generator that attaches to one edge of this 32×32 plane is now under way.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Bechenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

A 16-node Intel iPSC/2 was delivered in November 1988, and a 16-node Symult Series 2010, a second-generation medium-grain multicomputer developed as a

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

joint project between our research project and Symult Systems, Inc. (formerly Ametek Computer Research Division), was delivered in December 1988. Both of these systems have been used extensively for programming system developments, applications, and benchmarks. We have encountered very few system problems in running existing Cosmic-C application programs on either the Symult Series 2010 or Intel iPSC/2.

Application programs typical of those that were written for first-generation multicomputers run 8–10 times faster per node on the Symult Series 2010 and on the Intel iPSC/2 than on first-generation machines, such as the Intel iPSC/1. Applications involving latency-sensitive non-local message traffic exhibit more dramatic improvements, particularly on the Series 2010, due to cut-through message routing being included in the hardware of these second-generation multicomputers.

Delivery of a 64-node Series 2010 is expected on 31 March 1989, and our 16-node Series 2010 will be returned briefly to Symult to be upgraded to 32 nodes and retrofitted with some hardware improvements to the mesh termination and host interfaces. The 32-node Series 2010 will continue as our principal programming-system-development machine. The 64-node Series 2010 and the 16-node iPSC/2 will be made available to outside users through the Caltech Concurrent Supercomputing Facilities. Outside users will include researchers at Caltech, as well as those associated with the Rice-Caltech-Argonne-Los Alamos (NSF Science and Technology) Center for Research in Parallel Computation. These systems will also be available for use by researchers in the DARPA community; DARPA researchers should contact Chuck Seitz (`chuck@vlsi.caltech.edu`) to make arrangements for access.

We expect to expand both the Intel iPSC/2 and Symult Series 2010 to larger configurations by the early part of CY90.

Copies of the Cosmic Environment system have been distributed to 13 additional sites during this period, bringing the total copies distributed directly from the project to over 160.

An effort has been started to implement major extensions of the Cosmic Environment host runtime system and the Reactive Kernel node operating system. The new CE will be based internally on reactive programming, and will allow a more distributed management of a set of network-connected multicomputers. The extended RK will support global operations across sets of cohort processes, including barrier synchronization, sum, min, max, parallel prefix, and rank. Another extension will be the support of distributed data structures, such as sets and ordered sets. These new features will be implemented at the RK handler level, where the message latency is only a fraction of that at the protected user level. The implementation of these algorithms at the handler level permits the performance of global and distributed-data-structure operations in times that do not greatly exceed those of user-level operations dealing with single messages.

Our Caltech project continues to work with both Intel and Symult on the architectural design, message-routing methods and chips, and system software for medium-grain multicomputers. We expect to see additional major advances in the performance and programmability of these systems over the next two years. In addition, we continue to develop applications in VLSI design and analysis tools, and in other areas in which the programming of these multicomputer systems presents particular difficulties or opportunities.

2.3 Cosmic Cube Project

Wen-King Su, Jakov Seizovic, Chuck Seitz

The Cosmic Cubes that were built in our project in 1983 and the Intel iPSC/1 d7 that was contributed to the project in 1985 continue to operate very reliably. Overall usage has decreased somewhat with the appearance of the second-generation multicomputers, but the iPSC/1 continues to be used fairly heavily within the research group for discrete event simulations, and by Caltech students and faculty in Aeronautics for supersonic-flow computations.

Neither the 64-node or 8-node Cosmic Cubes exhibited any hard failures in this five-month period. The two original Cosmic Cubes have now logged 3.8 million node-hours with only four hard failures, three of which were chip failures in nodes, and one a power-supply failure. A node MTBF in excess of 1,000,000 hours is probable based on this reliability experience.

3. Concurrent Computation

3.1 Cantor

Nanette J. Boden, Chuck Seitz

Programming Fine-Grain Multicomputers

The experiments we reported previously in application programming using Cantor 2.0 and 2.2 have suggested a series of changes to the Cantor language. William C. Athas, who led the development of Cantor while he was a graduate student and post-doc in the project, and who is now at UT Austin, has incorporated these ideas into the definition of a new version of Cantor (3.0). The principal structural changes are the introduction of limited discretion in receiving messages according to type, and in the approach to implementing functions.

In developing the Cantor programming system for the Mosaic, we mean to allow for these changes so that we may change to Cantor 3.0 as soon as a new compiler is produced.

Cantor for the Mosaic

Development of Cantor runtime support for the Mosaic multicomputer has progressed significantly during the last five months. Initially, we defined a Cantor Abstract Machine (CAM) that represents an idealized machine for executing Cantor code. The CAM instruction set includes single instructions that encapsulate complicated Cantor operations, such as process creation and message passing. By design, the implementation of these operations can be varied within native code generators for experimenting with different strategies. With the Mosaic, for example, we use a macro-assembler that translates the implementation for each CAM instruction into Mosaic instructions.

The definition of the first version of the Cantor runtime system for the Mosaic consisted chiefly of freezing efficient implementations for process creation and message passing, and expressing them with Mosaic instructions. In the case of process creation, a software cache of available reference values is maintained on each node so that processes can be created with low latency. These reference values are later bound to actual processes by special *creator* processes located on each node that allocate memory for new processes. Receiving a message on the Mosaic is implemented by having the runtime system determine the destination process, and then run that process to absorb the message. The runtime system also communicates with the runtime systems on other nodes to manage resources within the node, *eg*, sending requests for more reference values to fill the software cache.

To evaluate different runtime system prototypes, we developed a Mosaic simulator that runs on existing medium-grain multicomputers, including the Cosmic

Cubes, Intel iPSCs, and the Symult 2010. A host program distributes the Mosaic code for a Cantor program to each simulated Mosaic node, and initiates computation by instantiating the *main* process of the Cantor program. Program output is achieved by instantiating a *console* process and passing its reference in messages.

Currently, our simulator is working on a test suite of simple Cantor programs. In the future, we plan to incorporate some of the more recent Cantor innovations, *eg*, functions and limited message discretion, into the simulator and into the runtime system. We are also planning experiments to evaluate different strategies for code distribution and memory allocation throughout Mosaic nodes.

3.2 Concurrent Logic Programming

Stephen Taylor

A commercially supported concurrent logic programming system was ported to our Symult Series 2010 multicomputer, and is available for all users of our project's multicomputers.

This system is composed of a compiler for the language *Strand*, and an environment for program development. The language provides an abstract message-passing framework for use in a variety of symbolic and system integration tasks. The system is also operational on Intel iPSC systems, networks of Suns, Mecho Transputer surfaces, PC Plug-in Transputer cards, Encore/Sequent shared memory machines, BBN Butterfly, and Atari personal machines. The system was used for a graduate course in compiler techniques this quarter, and will be used in a graduate course on concurrent programming in this coming quarter. It is also being used to study various applications in the *composition* research described in the following section of this report. Finally, a textbook describing the ideas embodied in the Strand system was recently completed, and will be published by Prentice-Hall in July 1989.

3.3 Programming with Composition

Mani Chandy, Stephen Taylor

We are interested in developing a notation for specifying concurrent algorithms and programs. Our goals are to support formal reasoning about program correctness and to provide efficient implementations of symbolic, numeric, and operating system codes. We have chosen *program composition* as a central notion due to its prevalence in both semantic models and program design methodologies.

During the past six months, we have considered the basic components of such a notation. Our conclusion is that there are four composition operators of importance. These operators are defined on program units; the method by which these units are implemented is relatively unimportant. It is natural to expect the notation to allow existing codes (written in Fortran, C, Lisp, Ada, *etc*) to be reused on

multicomputers. Moreover, the composition of these units will have a formal semantic characterization. To explore the utility of the notation, we are currently focussing on the hand compilation of non-trivial application codes. If performance results indicate that the notation is sufficiently efficient, we plan to build a compiler targeted to multicomputer architectures.

In the area of numeric computing we are studying a large fluid-flow problem developed in the department of Applied Mathematics at Caltech. This Fortran application computes the transition from a two-dimensional Taylor Vortex to three-dimensional wavy-vortex flow. Central to the application is a relaxation algorithm that employs a multigrid method. After benchmarking, we discovered that more than 70% of the execution time for the application was spent in the relaxation algorithm; thus, we decided to focus on this algorithm. Unfortunately, we arrived at a somewhat negative conclusion: The original algorithm was based on a sequential line-iteration scheme that afforded no opportunity for concurrent execution. As a result, we have converted the original code to use a point Gaussian relaxation algorithm; this appears more suitable. We are currently in the process of debugging a concurrent formulation of the algorithm.

In the area of symbolic computing we are studying a large automated reasoning program in conjunction with the Aerospace Corporation in Los Angeles. This program has been used extensively for checking the correctness of hardware specifications and Ada programs. A central component of the program is a *congruence closure* algorithm used for maintaining equality assertions. We began this research by investigating the opportunities for executing portions of this algorithm concurrently. This, again, led us to a somewhat negative conclusion: The granularity of typical invocations of the algorithm is too low to benefit from concurrent execution. We are now investigating a new algorithm that overlaps the execution of multiple equality assertions. Since a large number of these occur in a typical proof, we believe this to be a more suitable direction.

Finally, we are also interested in working with DNA sequencing programs, but have not yet made substantial progress in this area.

It should be understood that the objective of these application efforts is to test the utility of the program-composition notation, rather than to develop the applications themselves.

3.4 Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm

Wen-King Su, Chuck Seitz

During the past five months, additional simulations using the new logic simulator have been made, and a revision of the paper "Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm" (included as an appendix to this report) was written for publication in the 1989 SCS Eastern Multi-Conference. A

test version of the hybrid simulator has been implemented on top of the concurrent CMB variant simulators. Results from this prelimiary investigation are promising, and a new, more efficient version of the hybrid simulator is currently being written.

3.5 Distributed Snapshots

Mani Chandy

One of the fundamental problems in distributed systems appears trivial: Record the state of the system. The problem is, however, quite difficult because distributed systems do not have a single system-wide clock. If there were a clock, all processes could record their local states at a predetermined time. The problem of recording global states of distributed systems is at the core of a large number of problems in distributed systems, including deadlock detection, termination detection, and resource management. The paper, "The Essence of Distributed Snapshots," submitted to the *ACM Transactions on Computer Systems*, and included as an appendix to this report, presents necessary and sufficient conditions for a collection of local snapshots (recordings of local states) to be a global snapshot. The paper shows that many distributed algorithms can be developed in a systematic and straightforward manner from these conditions.

4. VLSI Design

4.1 The Design of the First Asynchronous Microprocessor

Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, Pieter J. Hazewindus

We have completed the design of an entirely asynchronous (self-timed, delay-insensitive) microprocessor. It is a 16-bit, RISC-like architecture with independent instruction and data memories. It has 16 registers, 4 buses, an ALU, and two adders. The size is about 20,000 transistors. Two versions have been fabricated: one in 2μm MOSIS SCMOS, and one in 1.6μm MOSIS SCMOS. (On the 2μm version, only 12 registers were implemented in order to fit the chip into the 84-pin 6600μm$\times$4600μm pad frame.)

With the exception of *isochronic forks* (see the paper included as an appendix to this report), the chips are entirely delay-insensitive, *ie*, their correct operation is independent of any assumption on delays in operators and wires except that the delays be finite. The circuits use neither clocks nor knowledge about delays.

The only exception to the design method is the interface with the memories. In the absence of available memories with self-timed interfaces, we have simulated the completion signal from the memories with an external delay. For testing purposes, the delay on the instruction memory interface is variable.

In spite of the presence of several floating n-wells, the 2μm version runs at 12 MIPS. The 1.6μm version runs at 18 MIPS. (Those performance figures are based on measurements from sequences of ALU instructions without carry. They do not take advantage of the overlap between ALU and memory instructions.) Those performance results are quite encouraging given that the design is very conservative: It uses static gates, dual-rail encoding of data, completion trees, *etc.*

Only two of the 12 2μm chips passed all tests, but 34 out of the 50 1.6μm chips were found to be entirely functional. However, within a certain range of values for the instruction memory delay, the 1.6μm version is not entirely functional. We cannot yet explain this phenomenon.

We have tested the chips under a wide range of VDD voltage values. At room temperature, the 2μm version is functional in a voltage range from 7V down to 0.35V! And it reaches 15 MIPS at 7V. We have also tested the chips cooled in liquid nitrogen. The 2μm version reaches 20 MIPS at 5V and 30 MIPS at 12V. The 1.6μm version reaches 30 MIPS at 5V. Of course, these measurements are made without adjusting any clocks (there are none), but simply by connecting the processor to a memory containing a test program and observing the rate of instruction execution. The results are summarized in Figure 1. The power consumption is 145mW at 5V, and 6.7mW at 2V. Figure 2 shows that the optimal *power-delay product* is obtained at 2V at room temperature.

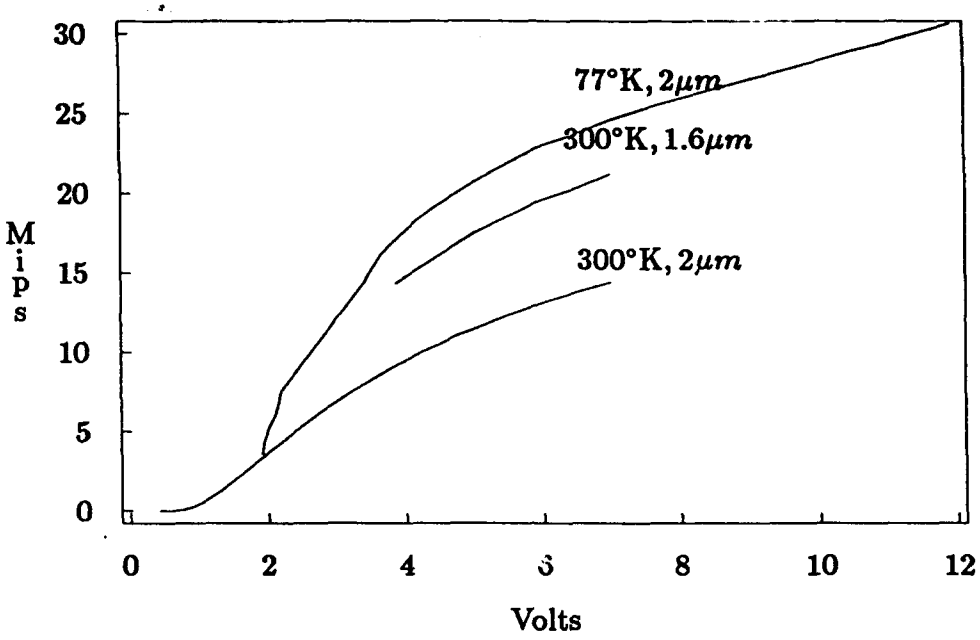


Figure 1: MIPS as a function of VDD

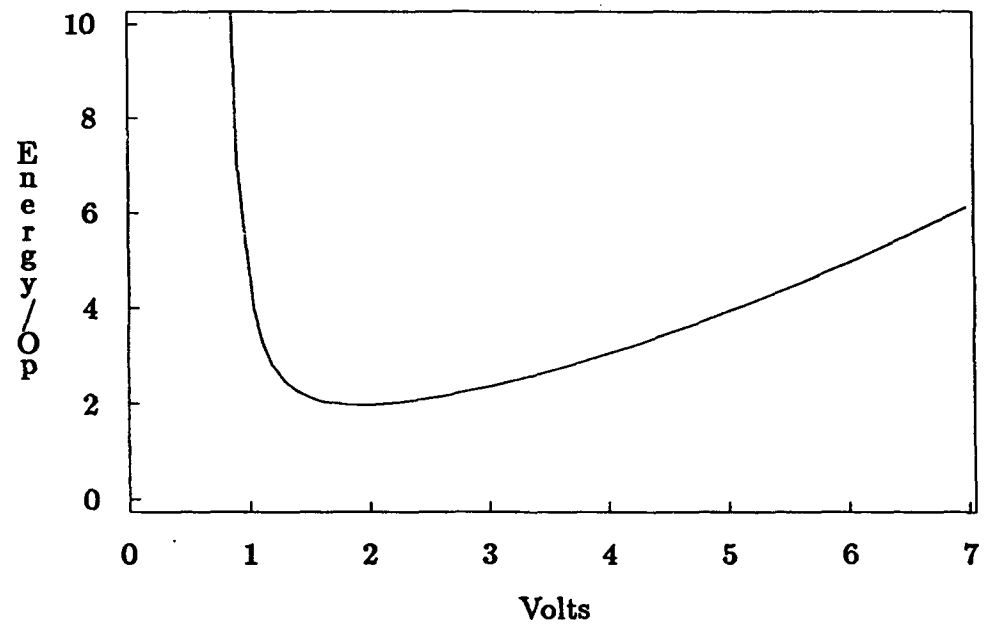


Figure 2: Power-delay product as a function of VDD

4.2 Fast Self-Timed Mesh Routing Chips

Chuck Seitz

The latest mesh-routing-chip (MRC) design, the FMRC2.1 design, was sent to MOSIS for 1.6μm SCMOS fabrication on 7 November 1988. This chip is a revision of FMRC2.0 that corrects a timing error in the latching of a routing decision. A Spice simulation indicated that that the revision corrected a timing error of approximately 0.7ns to a timing margin of about 1.0ns (about 50% of the difference between two short delay paths; hence, not as risky as it may sound). The maximal throughput predicted both by Spice and by tau-model calculations was 60MB/s.

These chips were returned from fabrication on 10 January 1989, and were found to operate correctly under a nearly exhaustive functional screening, and at a maximum throughput of 56MB/s. The yield on this run was 44/50. One of the chips had a cracked package, and two had bonding shorts; hence, the fabrication yield was actually 44/47.

Batches of 20 good chips were sent both to Intel Scientific Computers (as GFE on their DARPA contract) and to Symult Systems, and both companies have verified that these chips operate correctly in their test fixtures or systems.

The FMRC2.1 chip employs a design method that is *not* entirely delay-insensitive (see previous section). The circuit exhibits races *within* modules, but these modules have self-timed interfaces to other modules. Previous MRCs, entirely pin-for-pin compatible, employed the same delay-insensitive style as the asynchronous processor reported in the previous section, and required nearly twice the silicon area to operate half as fast as the FMRC2.1.

Hence, we conjecture that we shall see the same phenomenon with self-timed designs that is apparent with conventional designs; namely, that chips with relatively few cell types, such as memories and MRCs, will profitably employ circuit-level optimizations. Such optimizations are relatively less profitable and manageable in more complex chip designs, such as processors.

4.3 Mosaic C Chip

Jakov Seizovic, Jordan Holt, Chuck Seitz, Don Speck, Wen-King Su, Tony Wittry

During the past few months, work on the Mosaic chip has predominantly consisted of a series of extensive switch-level simulations. Using COSMOS instead of MOSSIM, we were able to decrease the simulation time by a factor of ten, with a negligible additional cost in setup (compile) time. The simulation of a memoryless version of Mosaic chip, consisting of about 26K transistors, takes slightly over a second of real time per clock cycle when running on a SUN 3/260. This has enabled us to simulate fairly long sequences of instructions from the Cantor runtime system at the switch-simulation level.

Having completed simulations of all of the logic parts of the Mosaic chip, *ie*, processor, packet interface, router, and bus arbiter, independently as well as together, we are entering the final phase of switch-level simulations, where multiple Mosaic chips will be represented as processes under CE/RK, and run on the multicomputers operated by the project, as well as on workstations.

We are planning to send the first version of a Mosaic chip to fabrication on a 2μ MOSIS run within a couple of weeks.

4.4 New CMOS PLAs

Jakov Seizovic, Chuck Seitz

A NOR-NOR precharged PLA has been designed to replace the NAND-NOR precharged PLA that we have used extensively since 1985. Both the delay and precharge time of this NOR-NOR PLA are linear in the number of inputs, a significant improvement compared to the NAND-NOR PLA, in which the delay is quadratic, and precharge time is cubic. This PLA has replaced the two NAND-NOR PLAs in the Mosaic C packet interface and the hybrid static/precharge NAND-NOR PLA in the Mosaic processor, and accordingly has saved us a lot of time and trouble in the Mosaic design.

4.5 CIF-flogger

Glenn Lewis, Chuck Seitz

CIF-flogger is a multicomputer program for flattening CIF files, rasterizing the geometry, and performing parallel operations on the geometry in strips. It runs under the CE/RK system, and hence, on most available multicomputers, including the Intel iPSC/2 and Symult Series 2010.

CIF-flogger currently supports the following operations on the chip geometry:

- parsing the CIF specification file (produced by Magic)
- flattening and rasterizing the hierarchical design geometry
- recognizing transistor geometry
- global connected-component labeling
- bloat, shrink, and logical mask layer operations
- creating new CIF for a processed design

Plans for CIF-flogger include:

- general CIF-reading capability
- circuit extraction

- well-plug checking
- design-rule checking

Initial timings indicate that CIF-flogger provides these operations in a matter of a few seconds for 100K-transistor chips. CIF-flogger is intended to be a useful tool for chip designers and foundries to verify that a design passes "syntactical" checks before it is fabricated, thus saving both time and money.

4.6 Adaptive Routing in Multicomputer Networks

John Y. Ngai, Chuck Seitz

As we are wrapping up our theoretical investigation of multicomputer adaptive routing, our recent efforts have been concentrated in two areas:

(1) The first of a series of publications will appear in the 1989 ACM Symposium on Parallel Algorithms and Architectures, to be held in Sante Fe, New Mexico this June. (A copy of this paper is included at the end of the report.)

(2) We have been searching for practical implementation ideas for replacing the existing oblivious router in the Mosaic with an adaptive router. A low-latency header encoding and modification scheme that we have dubbed the "sign-first one-shy code" has been devised for an adaptive router with a relatively narrow flit width. The details of these implementation ideas can be found in a forthcoming PhD thesis.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports

28 March 1989

Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check drawn on a U.S. bank in U.S. currency, payable to CALTECH.

CS-TR-89-03 $3.00 *Feature-oriented Image Enhancement with Shock Filters, I*
Rudin, Leonid I with Stanley Osher

CS-TR-89-02 $3.00 *Design of an Asynchronous Microprocessor,*
Martin, Alain J

CS-TR-89-01 $4.00 *Programming in VLSI From Communicating Processes to Delay-insensitive Circuits,*
Martin, Alain J

CS-TR-88-22 $2.00 *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm,*
Su, Wen-King and Charles L Seitz

CS-TR-88-21 $3.00 *Winner-Take-All Networks of O(N) Complexity,*
Lazzaro, John, with S Ryckebusch, M A Mahowald and C A Mead

CS-TR-88-20 $7.00 *Neural Network Design and the Complexity of Learning,*
Judd, J. Stephen

CS-TR-88-19 $5.00 *Controlling Rigid Bodies with Dynamic Constraints,*
Barzel, Ronen

CS-TR-88-18 $3.00 *Submicron Systems Architecture Project,*
ARPA Semiannual Technical Report

CS-TR-88-17 $3.00 *Constrained Differential Optimization for Neural Networks,*
Platt, John C and Alan H Barr

CS-TR-88-16 $3.00 *Programming Parallel Computers,*
Chandy, K Mani

CS-TR-88-15 $13.00 *Applications of Surface Networks to Sampling Problems in Computer Graphics,* PhD Thesis
Von Herzen, Brian

CS-TR-88-14 $2.00 *Syntax-directed Translation of Concurrent Programs into Self-timed Circuits*
Burns, Steven M and Alain J Martin

CS-TR-88-13 $2.00 *A Message-Passing Model for Highly Concurrent Computation,*
Martin, Alain J

CS-TR-88-12 $4.00 *A Comparison of Strict and Non-strict Semantics for Lists,* MS Thesis
Burch, Jerry R

CS-TR-88-11 $5.00 *A Study of Fine-Grain Programming Using Cantor,* MS Thesis
Boden, Nanette J

CS-TR-88-10 $3.00 *The Reactive Kernel,* MS Thesis
Seizovic, Jacov

CS-TR-88-07 $3.00 *The Hexagonal Resistive Network and the Circular Approximation,*
Feinstein, David I

CS-TR-88-06 $3.00 *Theorems on Computations of Distributed Systems,*
Chandy, K Mani

CS-TR-88-05 $3.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

CS-TR-88-04 $3.00 *Cochlear Hydrodynamics Demystified*
Lyon, Richard F and Carver A Mead

____CS-TR-88-03 $4.00 *PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics*, MS Thesis
Gupta, Rajiv

____CS-TR-88-02 $4.00 *Automated Compilation of Concurrent Programs into Self-timed Circuits*, MS Thesis
Steven M Burns

____CS-TR-88-01 $3.00 *C Programmer's Abbreviated Guide to Multicomputer Programming*,
Seitz, Charles, Jakov Seizovic and Wen-King Su

____5258:TR:88 $3.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5256:TR:87 $2.00 *Synthesis Method for Self-timed VLSI Circuits*,
Martin, Alain
current supply only: see *Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design*, 224-229, Oct'87

____5253:TR:88 $2.00 *Synthesis of Self-Timed Circuits by Program Transformation*,
Burns, Steven M and Alain J Martin

____5251:TR:87 $2.00 *Conditional Knowledge as a Basis for Distributed Simulation*,
Chandy, K. Mani and Jay Misra

____5250:TR:87 $10.00 *Images, Numerical Analysis of Singularities and Shock Filters*, PhD Thesis
Rudin, Leonid Iakov

____5249:TR:87 $6.00 *Logic from Programming Language Semantics*, PhD Thesis
Choo, Young-il

____5247:TR:87 $6.00 *VLSI Concurrent Computation for Music Synthesis*, PhD Thesis
Wawrzynek, John

____5246:TR:87 $3.00 *Framework for Adaptive Routing*
Ngai, John Y and Charles L. Seitz

____5244:TR:87 $3.00 *Multicomputers*
Athas, William C and Charles L Seitz

____5243:TR:87 $5.00 *Resource-Bounded Category and Measure in Exponential Complexity Classes*, PhD Thesis
Lutz, Jack H

____5242:TR:87 $8.00 *Fine Grain Concurrent Computations*, PhD Thesis
Athas, William C.

____5241:TR:87 $3.00 *VLSI Mesh Routing Systems*, MS Thesis
Flaig, Charles M

____5240:TR:87 $2.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5239:TR:87 $3.00 *Trace Theory and Systolic Computations*
Rem, Martin

____5238:TR:87 $7.00 *Incorporating Time in the New World of Computing System*, MS Thesis
Poh, Hean Lee

____5236:TR:86 $4.00 *Approach to Concurrent Semantics Using Complete Traces*, MS Thesis
Van Horn, Kevin S.

____5235:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5234:TR:86 $3.00 *High Performance Implementation of Prolog*
Newton, Michael O

____5233:TR:86 $3.00 *Some Results on Kolmogorov-Chaitin Complexity*, MS Thesis
Schweizer, David Lawrence

____5232:TR:86 $4.00 *Cantor User Report*
Athas, W.C. and C. L. Seitz

____5230:TR:86 $24.00 *Monte Carlo Methods for 2-D Compaction*, PhD Thesis
Mosteller, R.C.

____5229:TR:86 $4.00 *anaLOG - A Functional Simulator for VLSI Neural Systems*, MS Thesis
Lazzaro, John

5228:TR:86 $3.00 *On Performance of k-ary n-cube Interconection Networks*,
Dally, Wm. J

5227:TR:86 $18.00 *Parallel Execution Model for Logic Programming*, PhD Thesis
Li, Pey-yun Peggy

5223:TR:86 $15.00 *Integrated Optical Motion Detection*, PhD Thesis
Tanner, John E.

5221:TR:86 $3.00 *Sync Model: A Parallel Execution Method for Logic Programming*
Li, Pey-yun Peggy and Alain J. Martin
current supply only: see *Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86*

5220:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

5215:TR:86 $2.00 *How to Get a Large Natural Language System into a Personal Computer*,
Thompson, Bozena H. and Frederick B. Thompson

5214:TR:86 $2.00 *ASK is Transportable in Half a Dozen Ways*,
Thompson, Bozena H. and Frederick B. Thompson

5212:TR:86 $2.00 *On Seitz' Arbiter*,
Martin, Alain J

5210:TR:86 $2.00 *Compiling Communicating Processes into Delay-Insensitive VLSI Circuits*,
Martin, Alain
current supply only: see *Distributed Computing* v 1 no 4 (1986)

5207:TR:86 $2.00 *Complete and Infinite Traces: A Descriptive Model of Computing Agents*,
van Horn, Kevin

5205:TR:85 $2.00 *Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity*,
Schweizer, David and Yaser Abu-Mostafa

5204:TR:85 $3.00 *An Inverse Limit Construction of a Domain of Infinite Lists*,
Choo, Young-Il

5202:TR:85 $15.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5200:TR:85 $18.00 *ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation*, PhD thesis
Whelan, Dan

5198:TR:85 $8.00 *Neural Networks, Pattern Recognition and Fingerprint Hallucination*, PhD thesis
Mjolsness, Eric

5197:TR:85 $7.00 *Sequential Threshold Circuits*, MS thesis
Platt, John

5195:TR:85 $3.00 *New Generalization of Dekker's Algorithm for Mutual Exclusion*,
Martin, Alain J
current supply only: see *Information Processing Letters*, **23**, 295-297 (1986)

5194:TR:85 $5.00 *Sneptree - A Versatile Interconnection Network*,
Li, Pey-yun Peggy and Alain J Martin

5193:TR:85 $2.00 *Delay-insensitive Fair Arbiter*
Martin, Alain J

5190:TR:85 $3.00 *Concurrency Algebra and Petri Nets*,
Choo, Young-il

5189:TR:85 $10.00 *Hierarchical Composition of VLSI Circuits*, PhD Thesis
Whitney, Telle

5185:TR:85 $11.00 *Combining Computation with Geometry*, PhD Thesis
Lien, Sheue-Ling

5184:TR:85 $7.00 *Placement of Communicating Processes on Multiprocessor Networks*, MS Thesis
Steele, Craig

5179:TR:85 $3.00 *Sampling Deformed, Intersecting Surfaces with Quadtrees*, MS Thesis,
Von Herzen, Brian P.

5178:TR:85 $9.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

____5174:TR:85 $7.00 *Balanced Cube: A Concurrent Data Structure*,
Dally, William J and Charles L Seitz

____5172:TR:85 $6.00 *Combined Logical and Functional Programming Language*,
Newton, Michael

____5168:TR:84 $3.00 *Object Oriented Architecture*,
Dally, Bill and Jim Kajiya

____5165:TR:84 $4.00 *Customizing One's Own Interface Using English as Primary Language*,
Thompson, B H and Frederick B Thompson

____5164:TR:84 $13.00 *ASK French - A French Natural Language Syntax*, MS Thesis
Sanouillet, Remy

____5160:TR:84 $7.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

____5158:TR:84 $6.00 *VLSI Architecture for Sound Synthesis*,
Wawrzynek, John and Carver Mead

____5157:TR:84 $15.00 *Bit-Serial Reed-Solomon Decoders in VLSI*, PhD Thesis
Whiting, Douglas

____5147:TR:84 $4.00 *Networks of Machines for Distributed Recursive Computations*,
Martin, Alain and Jan van de Snepscheut

____5143:TR:84 $5.00 *General Interconnect Problem*, MS Thesis
Ngai, John

____5140:TR:84 $5.00 *Hierarchy of Graph Isomorphism Testing*, MS Thesis
Chen, Wen-Chi

____5139:TR:84 $4.00 *HEX: A Hierarchical Circuit Extractor*, MS Thesis
Oyang, Yen-Jen

____5137:TR:84 $7.00 *Dialogue Designing Dialogue System*, PhD Thesis
Ho, Tai-Ping

____5136:TR:84 $5.00 *Heterogeneous Data Base Access*, PhD Thesis
Papachristidis, Alex

____5135:TR:84 $7.00 *Toward Concurrent Arithmetic*, MS Thesis
Chiang, Chao-Lin

____5134:TR:84 $2.00 *Using Logic Programming for Compiling APL*, MS Thesis
Derby, Howard

____5133:TR:84 $13.00 *Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems*, PhD Thesis
Lin, Tzu-mu

____5132:TR:84 $10.00 *Switch Level Fault Simulation of MOS Digital Circuits*, MS Thesis
Schuster, Mike

____5129:TR:84 $5.00 *Design of the MOSAIC Processor*, MS Thesis
Lutz, Chris

____5128:TM:84 $3.00 *Linguistic Analysis of Natural Language Communication with Computers*,
Thompson, Bozena H

____5125:TR:84 $6.00 *Supermesh*, MS Thesis
Su, Wen-king

____5123:TR:84 $14.00 *Mossim Simulation Engine Architecture and Design*,
Dally, Bill

____5122:TR:84 $8.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

____5114:TM:84 $3.00 *ASK As Window to the World*,
Thompson, Bozena, and Fred Thompson

____5112:TR:83 $22.00 *Parallel Machines for Computer Graphics*, PhD Thesis
Ulner, Michael

____5106:TM:83 $1.00 *Ray Tracing Parametric Patches*,
Kajiya, James T

Caltech Computer Science Technical Reports

5104:TR:83 $9.00 *Graph Model and the Embedding of MOS Circuits*, MS Thesis
Ng, Tak-Kwong

5094:TR:83 $2.00 *Stochastic Estimation of Channel Routing Track Demand*,
Ngai, John

5092:TM:83 $2.00 *Residue Arithmetic and VLSI*,
Chiang, Chao-Lin and Lennart Johnsson

5091:TR:83 $2.00 *Race Detection in MOS Circuits by Ternary Simulation*,
Bryant, Randal E

5090:TR:83 $9.00 *Space-Time Algorithms: Semantics and Methodology*, PhD Thesis
Chen, Marina Chien-mei

5089:TR:83 $10.00 *Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits*,
Lin, Tzu-Mu and Carver A Mead

5086:TR:83 $4.00 *VLSI Combinator Reduction Engine*, MS Thesis
Athas, William C Jr

5082:TR:83 $10.00 *Hardware Support for Advanced Data Management Systems*, PhD Thesis
Neches, Philip

5081:TR:83 $4.00 *RTsim - A Register Transfer Simulator*, MS Thesis
Lam, Jimmy
current supply only: see *Acta Informatica* **20**, 301-313, (1983)

5074:TR:83 $10.00 *Robust Sentence Analysis and Habitability*,
Trawick, David

5073:TR:83 $12.00 *Automated Performance Optimization of Custom Integrated Circuits*, PhD Thesis
Trimberger, Steve

5065:TR:82 $3.00 *Switch Level Model and Simulator for MOS Digital Systems*,
Bryant, Randal E

5054:TM:82 $3.00 *Introducing ASK, A Simple Knowledgeable System*, Conf on App'l Natural Language Processing
Thompson, Bozena H and Frederick B Thompson

5051:TM:82 $2.00 *Knowledgeable Contexts for User Interaction*, Proc Nat'l Computer Conference
Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho

5035:TR:82 $9.00 *Type Inference in a Declarationless, Object-Oriented Language*, MS Thesis
Holstege, Eric

5034:TR:82 $12.00 *Hybrid Processing*, PhD Thesis
Carroll, Chris

5033:TR:82 $4.00 *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*,
Schuster, Mike, Randal Bryant and Doug Whiting

5029:TM:82 $4.00 *POOH User's Manual*,
Whitney, Telle

5018:TM:82 $2.00 *Filtering High Quality Text for Display on Raster Scan Devices*,
Kajiya, Jim and Mike Ullner

5017:TM:82 $2.00 *Ray Tracing Parametric Patches*,
Kajiya, Jim

5015:TR:82 $15.00 *VLSI Computational Structures Applied to Fingerprint Image Analysis*,
Megdal, Barry

5014:TR:82 $15.00 *Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, PhD Thesis
Lang, Charles R Jr

5012:TM:82 $2.00 *Switch-Level Modeling of MOS Digital Circuits*,
Bryant, Randal

5000:TR:82 $6.00 *Self-Timed Chip Set for Multiprocessor Communication*, MS Thesis
Whiting, Douglas

4684:TR:82 $3.00 *Characterization of Deadlock Free Resource Contentions*,
Chen, Marina, Martin Rem, and Ronald Graham

____4655:TR:81 $20.00 *Proc Second Caltech Conf on VLSI,*
Seitz, Charles, ed.

____3760:TR:80 $10.00 *Tree Machine: A Highly Concurrent Computing Environment*, PhD Thesis
Browning, Sally

____3759:TR:80 $10.00 *Homogeneous Machine*, PhD Thesis
Locanthi, Bart

____3710:TR:80 $10.00 *Understanding Hierarchical Design*, PhD Thesis
Rowson, James

____3340:TR:79 $26.00 *Proc. Caltech Conference on VLSI (1979),*
Seitz, Charles, ed

____2276:TM:78 $12.00 *Language Processor and a Sample Language,*
Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

Name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

____ Please check here if you wish to be included on our mailing list

____ Please check here for any change of address

____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

____89-03	____5250	____5210	____5143	____5074
____89-02	____5249	____5207	____5140	____5073
____89-01	____5247	____5205	____5139	____5065
____88-22	____5246	____5204	____5137	____5054
____88-21	____5244	____5202	____5136	____5051
____88-20	____5243	____5200	____5135	____5035
____88-19	____5242	____5198	____5134	____5034
____88-18	____5241	____5197	____5133	____5033
____88-17	____5240	____5195	____5132	____5029
____88-16	____5239	____5194	____5129	____5018
____88-15	____5238	____5193	____5128	____5017
____88-14	____5236	____5190	____5125	____5015
____88-13	____5235	____5189	____5123	____5014
____88-12	____5234	____5185	____5122	____5012
____88-11	____5233	____5184	____5114	____5000
____88-10	____5232	____5179	____5112	____4684
____88-07	____5230	____5178	____5106	____4655
____88-06	____5229	____5174	____5104	____3760
____88-05	____5228	____5172	____5094	____3759
____88-04	____5227	____5168	____5092	____3710
____88-03	____5223	____5165	____5091	____3340
____88-01	____5221	____5164	____5090	____2276
____5258	____5220	____5160	____5089	____....
____5256	____5215	____5158	____5086	____....
____5253	____5214	____5157	____5082	____....
____5251	____5212	____5147	____5081	____....

The Design of an Asynchronous Microprocessor

Alain J. Martin, Steven M. Burns, T.K. Lee,
Drazen Borkovic, Pieter J. Hazewindus

California Institute of Technology
Pasadena CA 91125, USA

to appear in *Proc. Decennial Caltech Conference on VLSI*, 20-22 March, 1989, MIT Press
Caltech-CS-TR-89-2

1 Introduction

Prejudices are as tenacious in science and engineering as in any other human activity. One of the most firmly held prejudices in digital VLSI design is that asynchronous circuits—a.k.a. self-timed or delay-insensitive circuits—are necessarily slow and wasteful in area and logic. Whereas asynchronous techniques would be appropriate for control, they would be inadequate for data paths because of the cost of dual-rail encoding of data, the cost of generating completion signals for write operations on registers, and the difficulty of designing self-timed buses.

Because a general-purpose microprocessor contains a complex data path, a corollary of the previous opinion is that it is impossible to design an efficient asynchronous microprocessor. Since we have been developing a design method for asynchronous circuits that gives excellent results, and since the above objections to large-scale data path designs are genuine but untested, we decided to "pick up the gauntlet" and design a complete processor.

The design of an asynchronous microprocessor poses new challenges and opens new avenues to the computer architect. Hence, the experiment unavoidably developed a dual purpose: We are refining an already well-tested design method, and we are starting a new series of experiments in asynchronous architectures. (As far as we know, this is the first entirely asynchronous microprocessor ever built.) The results we are reporting have a different implication depending on whether they are related to the first or second goal of the experiment. Whereas we are convinced that our design methods have reached maturity, we are quite aware that asynchronous techniques may influence the computer architects in completely new ways that this first design is just starting to explore.

In order to focus the experiment on asynchronous circuit design, we have intentionally excluded optimizations at the high and low ends of the design process. The instruction set is straightforward and no assumption has been made on the code produced by the compiler. No special electrical optimizations other than transistor sizing have been applied; the circuit techniques rarely go beyond those taught in a graduate-level VLSI class, and, apart from the memory interfaces, the circuits are ***delay-insensitive***. Hence, any performance is to be attributed to the design method and to the inherent advantages of asynchronous design.

A circuit is delay-insensitive when its correct operation is independent of any assumption on delays in operators and wires except that the delays be finite. Such circuits do not use a clock signal or knowledge about delays: Sequencing is enforced entirely by communication mechanisms.

The class of entirely delay-insensitive circuits is very limited. Different asynchronous techniques distinguish themselves in the choice of the compromises to delay-insensitivity. ***Speed-independent*** techniques assume that delays in gates are arbitrary, but there are no delays in wires. ***Self-timed*** techniques assume that a circuit can be decomposed into ***equipotential*** regions inside which delays in wires are negligible[11].

In our method, certain local forks are introduced to distribute a variable as inputs of several gates. We assume that the difference between the delays in the branches of such forks are short compared to delays in other gates. We call such forks ***isochronic***[6], [8].

The general method—a complete description of which can be found in the referenced papers [2], [5], [6], [7], [8]—is based on program transformations. The circuit is first designed as a set of concurrent programs. Each program is then compiled (manually or automatically) into a circuit by applying a series of program transformations. Control and data path are first designed separately and then combined in a mechanical way. This important ***divide-and-conquer*** technique is a main innovation of the method.

2 Preliminary Results

As of this writing, the first design is complete, and has been scheduled for fabrication in $2\mu m$ MOSIS SCMOS. The chip was functionally simulated using COSMOS [1], and was found to be functionally correct.

The architecture is a 16-bit processor with offset and a simple instruction set of the RISC type [4]. The data path contains twelve 16-bit registers, four buses, an ALU, and two adders. The chip contains 20,000 transistors and fits within a 5500λ by 3500λ area. We are using an 84-pin $6600\mu m \times 4600\mu m$ frame. An estimate of the critical path suggests processor performance of approximately 15MIPS in $2\mu m$ SCMOS. (A slightly improved $1.6\mu m$ SCMOS version is also being fabricated.)

This experiment, the most challenging one we have conducted so far, promised to be an important test for our method. The results obtained so far have been very encouraging.

The technique for separating control and data path has been extended with a novel asynchronous bus design, and is now robust and general.

The handshaking protocol between circuit elements has also been modified so that half of a protocol sequence overlaps subsequent actions. This protocol makes it possible to "hide" half of delays of the completion trees, the tree of gates that combine the completion signals from the asynchronous elements. In addition, at most two completion trees are in sequence on any path. Thus, completion tree delays are not a serious disadvantage of asynchronous design.

Instruction pipelining has been approached as a concurrent programming problem: Starting with a sequential program for the processor, concurrency is introduced through a series of program transformations. However, although the transformations are guided by the intent to overlap the important phases—fetch, decode, execute—of instruction execution, they are neither mechanical nor unique. The designer decides how to decompose a program into several concurrent ones. We do not claim that our solution in this first design is in any way optimal.

3 Specification of the Processor as a Sequential Program

The instruction set is deliberately not innovative. It is a conventional 16-bit-word instruction set of the *load-store* type. The processor uses two separate memories for instructions and data. There are three types of instructions: ALU, memory, and program-counter (*pc*). All ALU instructions operate on registers; memory instructions involve a register and a data memory word. Certain instructions use the following word as *offset*. (See Table 1 in Appendix 2.)

```
*[FETCH : i, pc := imem[pc], pc + 1;
          [offset(i.op) → offset, pc := imem[pc], pc + 1;
          |¬offset(i.op) → skip
          ];
 EXECUTE : [alu(i.op) → ⟨reg[i.x], f⟩ := aluf(reg[i.x], reg[i.y], i.op, f)
           |ld(i.op) → reg[i.x] := dmem[reg[i.x] + reg[i.y]]
           |st(i.op) → dmem[reg[i.x] + reg[i.y]] := reg[i.x]
           |ldx(i.op) → reg[i.x] := dmem[offset + reg[i.y]]
           |stx(i.op) → dmem[offset + reg[i.y]] := reg[i.x]
           |lda(i.op) → reg[i.x] := offset + reg[i.y]
           |stpc(i.op) → reg[i.x] := pc
           |jmp(i.op) → pc := reg[i.y]
           |brch(i.op) → [cond(f, i.cc) → pc := pc + offset
                         |¬cond(f, i.cc) → skip
                         ]
           ]
].
```

Figure 1: Sequential program describing the processor

The only important omissions, those of an interrupt mechanism and communication ports, are ones we found to be unnecessary distractions in a first design.

The sequential program describing the processor is a non-terminating loop, each step of which is a *FETCH* phase followed by an *EXECUTE* phase. The complete sequential program for the processor is shown in Figure 1. (The notation, which is an extension of the one we have used in previous work, is described in Appendix 1.) Variable i, which contains the instruction currently being executed, is described in the PASCAL record notation as a structured variable consisting of

several fields. All instructions contain an *op* field for the *opcode*. The parameter fields depend on the types of the instructions, which are found in Table 2 in Appendix 2. The most common ones, thôse for ALU, load, and store instructions, consist of the three parameters, x, y, and z. Variable *cc* contains the condition code field of the branch instruction, and *f* contains the *flags* generated by the execution of an *alu* instruction.

The two memories are the arrays *imem* and *dmem*. The index to *imem* is the program-counter variable, *pc*. The general-purpose registers are described as the array *reg*[0...15]. (Only twelve registers are implemented in the first chip.) Register *reg*[0] is special: It always contains the value zero.

4 Decomposition into Concurrent Processes

We decompose the previous program into a set of concurrent processes that communicate and synchronize using communication commands on channels. A restricted form of shared variables is allowed. The control channels *Xs*, *Ys*, *ZAs*, *ZWs*, *ZRs*, and the bus *ZA* are one-to-many; the buses *X*, *Y*, *ZM* are many-to-many; the other channels are one-to-one. But all channels are used by only two processes at a time. The structure of processes and channels is shown in Figure 2. The final program is shown in Figures 3 and 4.

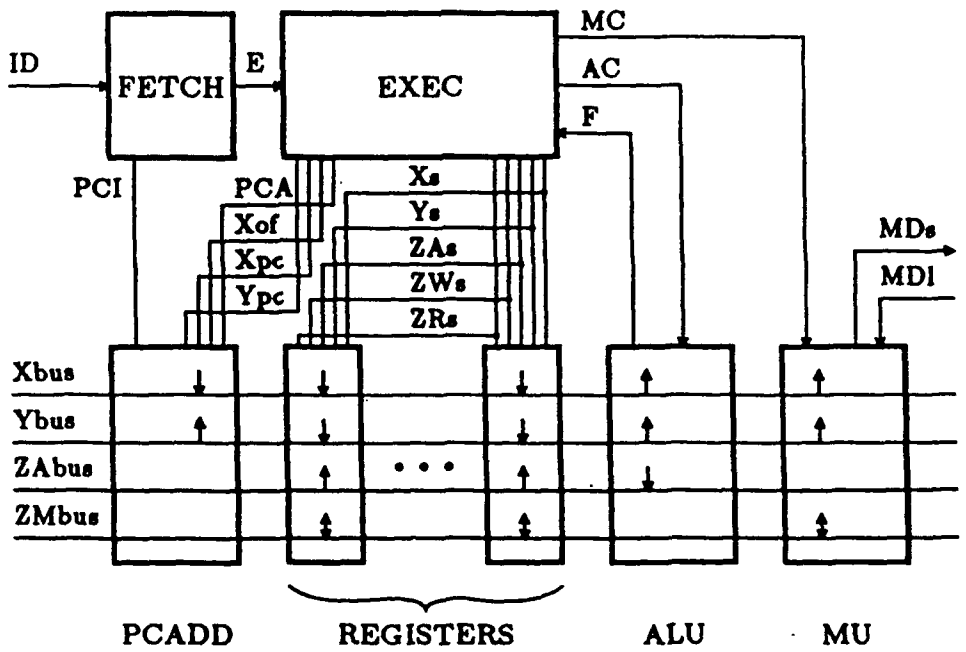


Figure 2: Process and channel structure

$$
\begin{aligned}
&IMEM \equiv *[ID!imem[pc]] \\
&FETCH \equiv *[PCI1; ID?i; PCI2; \\
&\quad [offset(i.op) \rightarrow PCI1; ID?offset; PCI2 \\
&\quad \|\neg offset(i.op) \rightarrow skip \\
&\quad]; E1!i; E2 \\
&\quad] \\
&PCADD \equiv (*[[\overline{PCI1} \rightarrow PCI1; y := pc + 1; PCI2; pc := y \\
&\quad \|\overline{PCA1} \rightarrow PCA1; y := pc + offset; PCA2; pc := y \\
&\quad \|\overline{Xpc} \rightarrow X!pc \bullet Xpc \\
&\quad \|\overline{Ypc} \rightarrow Y?pc \bullet Ypc \\
&\quad]] \\
&\quad \|*[[\overline{Xof} \rightarrow X!offset \bullet Xof]] \\
&\quad) \\
&EXEC \equiv *[E1?i; \\
&\quad [alu(i.op) \rightarrow E2; Xs \bullet Ys \bullet AC!i.op \bullet ZAs \\
&\quad \|ld(i.op) \rightarrow E2; Xs \bullet Ys \bullet MC1 \bullet ZRs \\
&\quad \|st(i.op) \rightarrow E2; Xs \bullet Ys \bullet MC2 \bullet ZWs \\
&\quad \|ldx(i.op) \rightarrow Xof \bullet Ys \bullet MC1 \bullet ZRs; E2 \\
&\quad \|stx(i.op) \rightarrow Xof \bullet Ys \bullet MC2 \bullet ZWs; E2 \\
&\quad \|lda(i.op) \rightarrow Xof \bullet Ys \bullet MC3 \bullet ZRs; E2 \\
&\quad \|stpc(i.op) \rightarrow Xpc \bullet Ys \bullet AC!add \bullet ZAs; E2 \\
&\quad \|jmp(i.op) \rightarrow Ypc \bullet Ys; E2 \\
&\quad \|brch(i.op) \rightarrow F?f; [cond(f, i.cc) \rightarrow PCA1; PCA2 \\
&\qquad\qquad\qquad\qquad \|\neg cond(f, i.cc) \rightarrow skip \\
&\qquad\qquad\qquad\qquad]; E2 \\
&\quad]
\end{aligned}
$$

Figure 3: The final program, first part

$$
\begin{aligned}
ALU \equiv *[[&\overline{AC} \rightarrow AC?op \bullet X?x \bullet Y?y; \\
&\qquad \langle z, f \rangle := aluf(x, y, op, f); ZA!z \\
&\;\; [\overline{F} \rightarrow F!f \\
&\;\;]] \\
MU \equiv *[[&\overline{MC1} \rightarrow X?x \bullet Y?y \bullet MC1; ma := x + y; MDl?w; ZM!w \\
&\;\; [\overline{MC2} \rightarrow X?x \bullet Y?y \bullet MC2 \bullet ZM?w; ma := x + y; MDs!w \\
&\;\; [\overline{MC3} \rightarrow X?x \bullet Y?y \bullet MC3; ma := x + y; ZM!ma \\
&\;\;]] \\
DMEM \equiv *[[&\overline{MDl} \rightarrow MDl!dmem[ma] \\
&\qquad [\overline{MDs} \rightarrow MDs?dmem[ma] \\
&\qquad]] \\
REG[k] \equiv (*[[&\neg bk \wedge k = i.x \wedge \overline{Xs} \rightarrow X!r \bullet Xs]] \\
&\;\; \| *[[\neg bk \wedge k = i.y \wedge \overline{Ys} \rightarrow Y!r \bullet Ys]] \\
&\;\; \| *[[\neg bk \wedge k = i.z \wedge \overline{ZWs} \rightarrow ZM!r \bullet ZWs]] \\
&\;\; \| *[[\neg bk \wedge k = i.z \wedge \overline{ZAs} \rightarrow bk\uparrow; ZAs; ZA?r; bk\downarrow]] \\
&\;\; \| *[[\neg bk \wedge k = i.z \wedge \overline{ZRs} \rightarrow bk\uparrow; ZRs; ZM?r; bk\downarrow]] \\
&\;\;)
\end{aligned}
$$

Figure 4: The final program, second part

Process *FETCH* fetches the instructions from the instruction memory, and transmits them to process *EXEC* which decodes them. Process *PCADD* updates the address *pc* of the next instruction concurrently with the instruction fetch, and controls the *offset* register. The execution of an ALU instruction by process *ALU* can overlap with the execution of a memory instruction by process *MU*. The *jump* and *branch* instructions are executed by *EXEC*; *store-pc* is executed by the ALU as the instruction "add the content of register r to the pc and store it." The array *REG*[*k*] of processes implements the register file. Both *MU* and *PCADD* contain their own adder. Processes *IMEM* and *DMEM* describe the instruction memory and data memory, respectively.

Updating the PC

The variable *pc* is updated by process *PCADD*, and is used by *IMEM* as the index of the array *imem* during the *ID* communication—the instruction fetch.

The assignment *pc* := *pc*+1 is decomposed into *y* := *pc*+1; *pc* := *y*, where *y* is a local variable of *PCADD* . The overlap of the instruction fetch, *ID*? (either *ID*?*i* or *ID*?*offset*), and the pc increment, *y* := *pc* + 1, can now occur while *pc* is constant. Action *ID*? is enclosed between the two communication actions *PCI1* and *PCI2*, as follows:

$$PCI1; ID?i; PCI2 .$$

In *PCADD*, *y* := *pc* + 1 is enclosed between the same two communication actions while the updating of *pc* follows *PCI2*:

$$\overline{PCI1} \rightarrow PCI1; y := pc + 1; PCI2; pc := y .$$

Since the completions of *PCI1* and *PCI2* in *FETCH* coincide with the completion of *PCI1* and *PCI2* in *PCADD*, respectively, the execution of *ID*?*i* in *FETCH* overlaps the execution of *y* := *pc* + 1 in *PCADD*. *PCI1* and *PCI2* are implemented as the two halves of the same communication handshaking to minimize the overhead.

In order to concentrate all increments of *pc* inside *PCADD*, we use the same technique to delegate the assignment *pc* := *pc* + *offset* (executed by the *EXEC* part in the sequential program) to *PCADD*.

The guarded command $\overline{Xof} \rightarrow Xof!offset$ in *PCADD* has been transformed into a concurrent process since it needs only be mutually exclusive with assignment *y* := *x* + *offset*, and this mutual exclusion is enforced by the sequencing between *PCA1*; *PCA2* and *Xof* within *EXEC*.

5 Stalling the Pipeline

When the pc is modified by *EXEC* as part of the execution of a *pc* instruction, (*store-pc*, *jump* or *branch*), fetching the next instruction by *FETCH* is postponed until the correct value of the pc is assigned to *PCADD.pc*.

When the offset is reserved for *MU* by *EXEC*, as part of the execution of some memory instructions, fetching the next instruction, which might be a new offset, is postponed until *MU* has received the

value of the current offset. In the second design, we have refined the protocol to block *FETCH* only when the next instruction is a new offset.

Postponing the start of the next cycle in *FETCH* is achieved by postponing the completion of the previous cycle, i.e., by postponing the completion of the communication action on channel E. As in the case of the *PCI* communication, E is decomposed into two communications, $E1$ and $E2$. Again, $E1$ and $E2$ are implemented as the two halves of the same handshaking protocol.

In *FETCH*, $E!i$ is replaced with $E1!i; E2$. In *EXEC*, $E2$ is postponed until after either *Xof?offset* or a complete execution of a *pc* instruction has occurred.

6 Sharing Registers and Buses

A bus is used by two processes at a time, one of which is a register and the other is *EXEC*, *MU*, *ALU*, or *PCADD*. We therefore decided to introduce enough buses so as not to restrict the concurrent access to different registers. For instance, *ALU* writing a result into a register should not prevent *MU* from using another register at the same time.

The four buses correspond to the four main concurrent activities involving the registers.

The X bus and the Y bus are used to send the parameters of an ALU operation to the ALU, and to send the parameters of address calculation to the memory unit. We also make opportunistic use of them to transmit the pc and the offset to and from *PCADD*.

The ZA bus is used to transmit the result of an ALU operation to the registers. The ZM bus is used by the memory unit to transmit data between the data memory and the registers.

We make a virtue out of necessity by turning the restriction that registers can be accessed only through those four buses into a convenient abstraction mechanism. The ALU uses only the X, Y, and ZA ports without having to reference the particular registers that are used in the communications. It is the task of *EXEC* to reserve the X, Y, and ZA bus for the proper registers before the ALU uses them.

The same holds for the *MU* process, which references only X, Y, and ZM. An additional abstraction is that the X bus is used to send the offset to *MU*, so that the cases for which the first parameter is $i.x$ or *offset* are now identical, since both parameters are sent via the X bus.

Exclusive Use of a Bus

Commands *Xpc*, *Ypc*, and *Xof* are used by *EXEC* to select the X and Y buses for communication of *pc* and *offset*. Commands *Xs*, *Ys*, and *ZAs* are used by *EXEC* to select the X, Y, and ZA buses, respectively, for a register that has to communicate with the ALU as part of the execution of an ALU instruction.

Two commands are needed to select the ZM bus: *ZWs* if the bus is to be used for writing to the data memory, and *ZRs* if the bus is to be used for reading from the data memory.

Let us first solve the problem of the mutual exclusion among the different uses of a bus. As long as we have only one ALU and one memory unit, no conflict is possible on the ZA and ZM buses, since only the ALU uses the ZA bus, and only the memory unit uses the ZM bus. But the X and Y buses are used concurrently by the ALU, the memory unit, and the pc unit.

We achieve mutual exclusion on different uses of the X bus as follows. (The same argument holds for Y.) The completion of an X communication is made to coincide with the completion of one of the selection actions *Xs*, *Xof*, *Xpc*; and the occurrences of these selection actions exclude each other in time inside *EXEC* since they appear in different guarded commands.

This coincidence is implemented by the *bullet* ($\bullet$) command : For arbitrary communication commands U and V inside the same process, $U \bullet V$ guarantees that the two actions are completed at the same time. We then say that the two actions coincide. The use of the bullets *X!pc* $\bullet$ *Xpc* and *X!offset* $\bullet$ *Xof* inside *PCADD* , and *X!r* $\bullet$ *Xs* inside the registers enforce the coinidence of X with *Xpc*, *Xof*, and *Xs*, respectively. The bullets in *EXEC*, *ALU*, and *MU* have been introduced for reasons of efficiency: Sequencing is avoided.

7 Register Selection

Command *Xs* in *EXEC* selects the X bus for the particular register whose index k is equal to the field $i.x$ of the instruction i being decoded by *EXEC*, and analogously for commands *Ys*, *ZAs*, *ZRs*, and *ZWs*.

Each register process *REG*[k], for $0 \leq k < 16$, consists of five elementary processes, one for each selection command. The register that is selected by command *Xs* is the one that passes the test $k = i.x$. This implementation requires that the variable $i.x$ be shared by all

registers and *EXEC*. An alternative solution that does not require shared variables uses demultiplexer processes. (The implementations of the two solutions are almost identical.)

The semicolons in the last two guarded commands of $REG[k]$ are introduced to pipeline the computation of the result of an ALU instruction or memory instruction with the decoding of the next instruction.

Mutual Exclusion on Registers

A register may be used in several arguments (x, y, or z) of the same instruction, and also as an argument in two successive instructions whose executions may overlap. We therefore have to address the issue of the concurrent uses of the same register. Two concurrent actions on the same register are allowed when they are both read actions.

Concurrency within an instruction is not a problem: X and Y communications on the same register may overlap, since they are both read actions, and Z cannot overlap with either X or Y because of the sequencing inside *ALU* and *MU*.

Concurrency in the access to a register during two consecutive overlapping instructions (one instruction is an ALU and the other is a memory instruction) can be a problem: Writing a result into a register (a ZA or a ZR action) in the first instruction can overlap with another action on the same register in the second instruction. But, because the selection of the z register for the first instruction takes place before the selection of the registers for the second instruction, we can use this ordering to impose the same ordering on the different accesses to the same register when a ZA or ZR is involved.

This ordering is implemented as follows: In $REG[k]$, variable bk (initially false) is set to true before the register is selected for ZA or ZR, and it is set back to false only after the register has been actually used. All uses of the register are guarded with the condition $\neg bk$. Hence, all subsequent selections of the register are postponed until the current ZA or ZR is completed.

We must ensure that bk is not set to true before the register is selected for an X or a Y action ***inside the same instruction***, since this would lead to deadlock. We omit this refinement which does not appear in the program of Figures 3 and 4.

8 Implementation

Control Part

The control part of a process is obtained by the following transformations: First, each communication command involving message input or output is replaced with a "bare" communication on the channel; for instance, $C?x$ and $C!x$ would both be replaced with C.

Second, all assignment statements are delegated to subprocesses. Assignment S is replaced with a communication command on a new channel, say Cs, and the subprocess $*[[\overline{Cs} \rightarrow S \bullet Cs]]$ is introduced. After these transformations, the control part of each process consists only of boolean expressions in conditionals and of communication commands. Thus, the next step is to implement each communication command with a ***handshaking protocol.***

Handshaking Protocols

Consider the matching pair of actions $X!u$ and $X?v$ in processes A and B respectively. We first implement the bare communication on channel X. The channel is implemented by the two handshake wires $(xo \underline{w} yi)$ and $(yo \underline{w} xi)$ as indicated on Figure 5.(a). As usual, we use a four-phase, or "return-to-zero" handshaking protocol. Such a protocol is not symmetrical: All communications in one process are implemented as ***active*** and all communications in the other process as ***passive.***

We have shown in [7] and [8] that the implementation of an input action is significantly simpler when combined with an active protocol than with a passive one. Therefore all input actions are implemented as active and all output actions as passive. (In the case of output, the implementation of communication is the same for active and passive protocols.)

The standard active and passive implementations are:

$$[yi]; yo\uparrow; [\neg yi]; yo\downarrow \quad \text{(passive)}$$

$$xo\uparrow; [xi]; xo\downarrow; [\neg xi] \quad \text{(active)}\ .$$

(The passive protocol starts with the wait action $[yi]$, i.e., "wait until the input wire is set to true." The active protocol starts with $xo\uparrow$, i.e., "set the output wire to true.")

We introduce an alternative active implementation, called *lazy active*:

$$[\neg xi]; xo\uparrow; [xi]; xo\downarrow \quad \textbf{(lazy active)} .$$

The lazy active protocol differs from the active one in that the last wait action $[\neg xi]$ is postponed until the beginning of the next communication. The difference is important when data communication is involved.

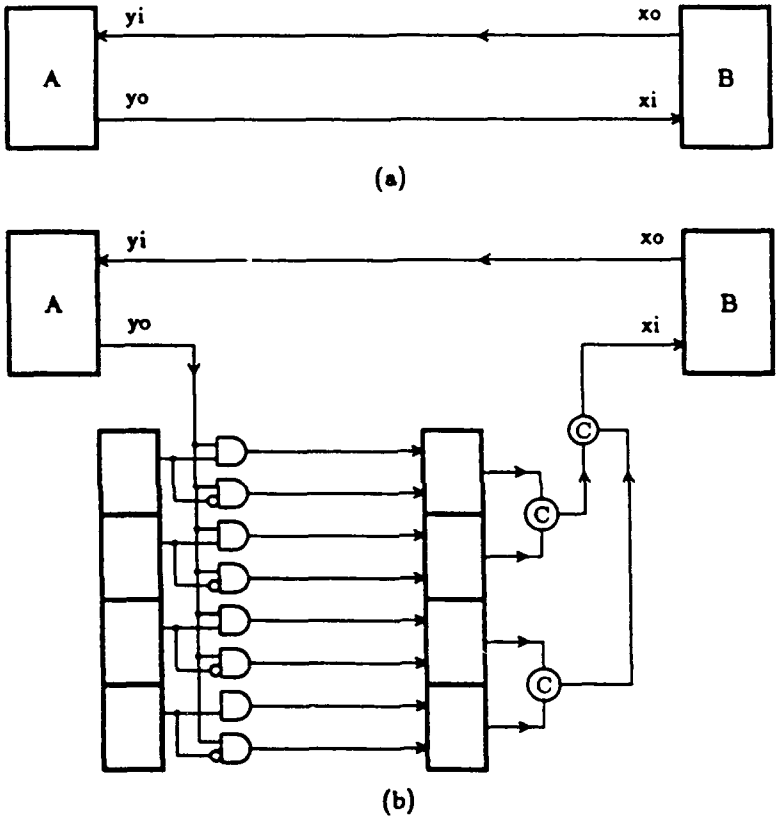


Figure 5: Implementation of communication

Figure 5.(b) shows how the data path is combined with the control. The bits of the communication channel between the two registers (the "data wires") are dual-rail encoded. Wire $(yo \underline{w} xi)$ is "cut open," yo is used to assigned the values of the bits of u to the dual-rail data wires, and xi is set to true when all bits of v have been set to the values of the data wires. Each cell of a register contains an acknowledge wire that is set to true when the bit of the cell has been set to a valid value of the two data wires, and reset to false when the data wires are both

reset to false. Let $vack_i$ be the acknowledge of bit v_i, xi is set and reset as:

$$vack_0 \wedge vack_1 \ldots \wedge vack_{15} \mapsto xi\uparrow$$
$$\neg vack_0 \wedge \neg vack_1 \ldots \wedge \neg vack_{15} \mapsto skip$$

Since a 16-input C-element would be prohibitively slow to implement, the implementation is a tree of smaller C-elements, which we call a ***completion tree***. Figure 5.(b) shows a tree of binary C-elements. In the actual processor, we use a two-level tree of 4-input C-elements.

When data is transmitted via a bus, and when the completion tree is large, the gain of using a lazy-active protocol can be very important, since half of the data transmission delays and half of the completion-tree delays can overlap with the rest of the computation. Therefore, all input actions are implemented as lazy active.

The case when data is transmitted from process A to process B via a bus is only slightly more complicated. No arbitration is necessary: A and B are allowed to communicate via a bus only after the bus has been reserved for these two processes. The chief problem in implementing the buses is the distributed implementation of large multi-input OR-gates.

The lazy-active protocol cannot be used when an input action is probed—such as action $AC?op$ in the ALU—because the probe requires a passive protocol. For those cases, we have designed a special protocol that requires two control wires.

9 ALU

ALU control

In the ALU process, variable z is not needed to store the result of an ALU operation: the result can be put directly on the ZA bus. The first guarded command of the ALU process can be rewritten:

$$\overline{AC} \rightarrow AC?op \bullet X?x \bullet Y?y; (ZA, f) := aluf(x, y, op, f).$$

Hence, the control part is simply:

$$*[[\overline{AC} \rightarrow AC \bullet X \bullet Y; AL$$
$$[]\overline{F} \rightarrow F$$
$$]].$$

(The assignment to f is omitted.) Communication command AL is the call of the subprocess evaluating ***aluf***. The handshaking protocol of AL is passive because it includes an output action on the ZA bus: $[ali]; alo\uparrow; [\neg ali]; alo\downarrow$. Hence, $alo\uparrow$ is the "go" signal for the ALU computation proper.

The first guarded command has the structure of a canonical stage of the pipeline. Parameters are simultaneously received on a set of ports, and the result is sent on another port as in:

$$*[L?x; R!f(x)].$$

Such a process is called a ***buffer***. Since L is implemented as lazy active, and R as passive, it is a ***lazy-active/passive buffer***. In the second design, where we have decomposed both the ALU and the memory processes into two processes in order to improve the pipeline, each stage of the pipeline is a lazy-active/passive buffer.

ALU data path

The output Z of the subprocess is dual-rail encoded. When the subprocess is called, variables x, y, and op have stable and valid values. Moreover, the content of op has been encoded in a KPG ("kill, propagate, generate") form which is used to produce the carry-out for each bit, and also for the result. The length of the carry chain is variable, which is an advantage in a fully asynchronous execution.

Since the carry-out of each bit is inverted relative to the carry-in, we alternate the logic encoding of the stages in the carry chain: A carry-in that has a true value when high generates a carry-out that has a true value when low, and vice-versa for the next stage. With this coding, only one CMOS gate delay is incurred per stage. Although the acknowledge from the ZA bus is used as completion signal, a completion tree is needed at the output of the subprocess for the computation of the flags.

The elapsed time between the activation of the ALU subprocess by $alo\uparrow$ and the appearance of the results on the output Z depends on the number of stages in the carry chain. Add, substract, and other logical functions typically take between 13 and $25ns$ in $2\mu m$ SCMOS.

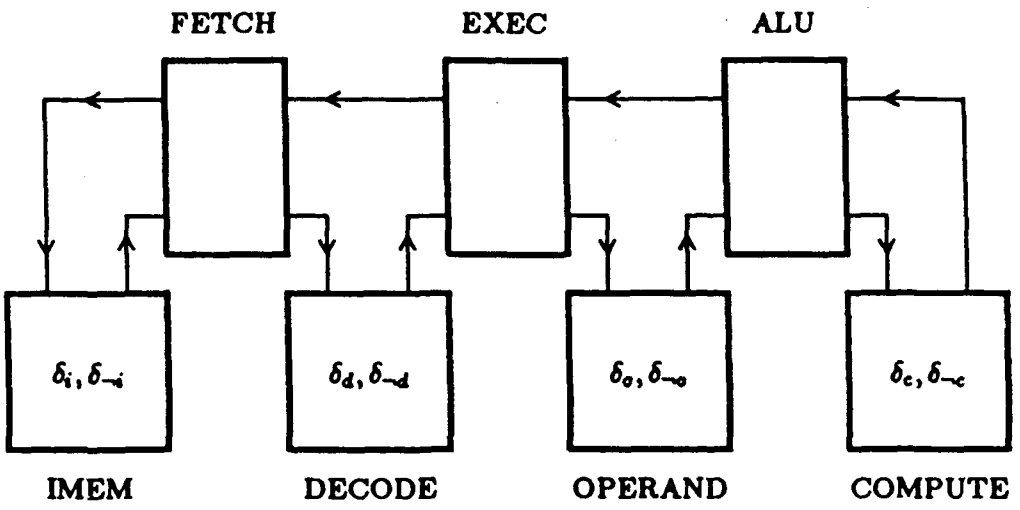


Figure 6: Abstract Pipeline for ALU Instructions

10 Performance

In this processor, an instruction is executed in a varying amount of time, depending in part on the type of instruction and the values of its operands, and on the sequence surrounding the instruction. Because of this data dependence, an analysis of the "real" performance of the processor, i.e., the performance of the processor when executing "real" programs, is quite complex and most probably must be determined by simulation. The performance analysis can be simplified by assuming an infinite sequence of identical instructions with typical operand values. (The results obtained through this analysis do not include the potential benefits of interleaving ALU and memory instructions.) Here, we analyze the performance of the processor executing an infinite sequence of ALU instructions.

In this case, the processor can be viewed as the three-stage pipeline shown in Figure 6. By assuming the ALU operations are performed on distinct registers, the register locking mechanism need not be introduced and the control for the *EXEC* process and the *ALU* process reduces to lazy-active/passive buffers. The *fetch* process is complicated by the increment of the *pc*, but if the instruction memory is assumed to be slower than the increment, control for this process also reduces to a lazy-active/passive buffer. By first assuming negligible control delays compared with datapath delays (denoted δ_D and $\delta_{\neg D}$ for the upgoing and downgoing propagation delays of datapath unit D, respectively),

the cycle time, c_P, of each process P is determined by the datapath delays that must be sequenced. A lazy-active/passive buffer sequences only the upgoing transitions of the two datapath units and, separately, the upgoing and downgoing transitions of the individual units, resulting in cycle time $\max(\delta_{D1} + \delta_{D2}, \delta_{D1} + \delta_{\neg D1}, \delta_{D2} + \delta_{\neg D2})$.

Since each process in the pipeline is a lazy-active/passive buffer, and since the throughput of the pipeline is determined by the slowest process:

$$c_{FETCH} = \max(\delta_m + \delta_d, \delta_m + \delta_{\neg m}, \delta_d + \delta_{\neg d})$$
$$c_{EXEC} = \max(\delta_d + \delta_o, \delta_d + \delta_{\neg d}, \delta_o + \delta_{\neg o})$$
$$c_{ALU} = \max(\delta_o + \delta_c, \delta_o + \delta_{\neg o}, \delta_c + \delta_{\neg c})$$
$$c_{PROC} = \max(c_{FETCH}, c_{EXEC}, c_{ALU}) .$$

Timing simulations suggest that the dominant constraints are the memory and decode sequence in the ***FETCH*** process ($\delta_m + \delta_d$), and the operand and compute sequence in the ALU process ($\delta_o + \delta_c$). For the $2\mu m$ SCMOS processor, the delays introduced by the control parts increase the cycle time by 10 to $20ns$, bringing the cycle time for an infinite stream of ALU instructions up to $\max(35ns + \delta_m, 65ns)$. We expect the processor to achieve 15 MIPS if the access delay of the instruction memory (δ_m) is no longer than $30ns$.

11 Correctness by Construction and CAD Tools

Since the method is based on semantics-preserving program transformations, the object code generated by the compilation procedure is correct by construction.

The object code is a set of potentially concurrent ***production rules*** that are constructs of the form $B1 \mapsto x\uparrow$ or $B2 \mapsto x\downarrow$, where $B1$ and $B2$ are mutually exclusive boolean expressions, and $x\uparrow$ and $x\downarrow$ stand for "set x to true" and "set x to false," respectively. The compilation procedure guarantees the absence of hazards by ensuring that the conditions $B1$ and $B2$ are ***stable***, i.e., if $B1$ is true, it remains true until x as been set to true.

If the production rules of the object code can be matched with the production rules that describe the standard cells of a cell library, a standard-cell-layout program can be used to generate a layout corresponding to the object code. We have been using such a standard cell approach in our previous designs, and indeed all chips fabricated in this way have been found to be functional on "first silicon."

However, most of the processor was designed manually. First, since the control section introduces significant overhead, we decided to

compile its object code manually. Second, because the data path was expected to be the critical part with respect to size and because of the difficulty of adjusting the pitch of the different registers automatically, the automatic layout program was used for the control part but not for the data path. This decision was later justified by the fact that, whereas the data path was hardly changed after the first design, the control part went through a series of drastic modifications. We observed that, again, our method for separating control and data path permitted us to implement completely different pipelines by changing the control without significant alterations of the data path.

As usual, the disadvantage of manual compilation was that the design was not shielded from clerical errors at which humans excel.

While the difficult optimization problem that is at the core of a high-performance processor design is probably still beyond automatic compilation technology, the designer should be assisted with CAD tools that perform the mechanical translation steps. Other CAD tools that we found useful include a program that estimates the critical path of a circuit. The program, which was developed by Steve Burns, gives excellent results. It estimates the delays of each path by a simulation of the execution based on the production rules.

Magic was used for the manual layout [10].

12 Conclusion

Although the chips are still in fabrication, we are very satisfied with the preliminary results of the experiment.

First, the chip layout is obviously not large. The control is surprisingly small despite our use of an automatic layout tool; also, the anticipated nightmare of data path layout did not materialize. The register pitch is 80λ, which is quite reasonable given that four buses have to be placed.

Second, the predicted performance is quite remarkable, given that the experiment is a first in two ways: It is our first experience as computer architects, and it is the first asynchronous microprocessor ever built.

Third, the complete design took five persons (one joined in the middle of the project) five months.

Since the choice of an instruction set was not part of the experiment, our design should be judged in two ways: the choice of the concurrent program of Figure 3, and its implementation.

The implementation is satisfactory, but not optimal. The sizing of transistors can be improved and the number of transitions can be decreased, mainly by a better placement of inverters. For instance, the delays due to a completion tree and to the control for a buffer are both about twice their theoretical minimum.

The program of Figure 3 represents the choice of a pipeline, and of synchronization techniques to implement it. We have deliberately chosen a simple pipeline. In particular, the mechanism for stalling, which places part of the decoding in series with the fetch on the critical path, sacrifices efficiency for simplicity. However, performance evaluations show that the pipeline is well-balanced since the different stages have comparable average delays. Improving the critical path by overlapping fetch and decode requires improving the ALU and memory instruction execution stages by pipelining parts of these stages.

The practicality of overlapping ALU and memory instruction executions remains an open issue. It is not clear whether the gain in performance is worth the complexity of the synchronization involved and the requirement of two separate Z buses.

We find the synchronization techniques used to implement the concurrent activities between the different stages of the pipeline particularly elegant and efficient, since the delays incurred in a synchronization can be of arbitrary length and vary from instruction to instruction.

We foresee excellent performances for asynchronous processors as the feature size keeps decreasing. But the designer must be ready to learn and apply new design methods based on concurrent programmming, that are required to exploit asynchronous techniques to their fullest.

Acknowledgment

We are indebted to Bill Athas and Bill Dally for useful discussions in the preliminary stage of the design. Chuck Seitz, Nanette Boden, and Dian De Sha made excellent comments on the manuscript. The first author enjoyed numerous discussions with Chuck Seitz on the general topic of asynchronous design.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, DARPA Order numbers 3771 & 6202, and monitored by the Office of Naval Research under contract numbers N00014-79-C-0597 & N00014-87-K-0745.

References

[1] R.E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler. COSMOS: A Compiled Simulator for MOS Circuits. In *24th Design Automation Conference*, pp.9–16. ACM and IEEE, 1987.

[2] Steven M. Burns and Alain J. Martin, Syntax-directed Translation of Concurrent Programs into Self-timed Circuits. In J. Allen and F. Leighton (ed), *Fifth MIT Conference on Advanced Research in VLSI,* pp 35-40, MIT Press, 1988.

[3] C.A.R. Hoare, Communicating Sequential Processes. *Comm. ACM* 21,8, pp 666-677, August, 1978.

[4] Mark Horowitz *et al.*, MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache. *IEEE Journal of Solid-State Circuits*,SC-22(5):790-799,October, 1987.

[5] Alain J. Martin, The Design of a Self-timed Circuit for Distributed Mutual Exclusion. In Henry Fuchs (ed), *1985 Chapel Hill Conf. VLSI*, Computer Science Press, pp 247-260, 1985.

[6] Alain J. Martin, Compiling Communicating Processes into Delay-insensitive VLSI Circuits. *Distributed Computing*, 1,(4), Springer-Verlag, pp 226-234 1986.

[7] Alain J. Martin, A Synthesis Method for Self-timed VLSI Circuits. *ICCD 87: 1987 IEEE International Conference on Computer Design*, IEEE Computer Society Press, pp 224-229, 1987.

[8] Alain J. Martin, Programming in VLSI: From Communicating Processes to Delay-insensitive Circuits. In C.A.R. Hoare (ed), *UT Year of Programming Institute on Concurrent Programming*, Addison-Wesley, Reading MA, 1989.

[9] Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading MA, 1980.

[10] J. K. Ousterhout *et al.*, The Magic VLSI layout system, *IEEE Design Test Comput.*, 2, (1), pp 19-30, February, 1985.

[11] Charles L. Seitz, System Timing, Chapter 7 in Mead & Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading MA, 1980.

Appendix 1: Notation

The program notation, which is inspired by C.A.R. Hoare's CSP [3], is briefly described.

$b\uparrow$ stands for $b :=$ **true**, $b\downarrow$ stands for $b :=$ **false**.

The execution of the ***selection*** command $[G_1 \rightarrow S_1 \| \ldots \| G_n \rightarrow S_n]$, where G_1 through G_n are boolean expressions, and S_1 through S_n are program parts, (G_i is called a "guard," and $G_i \rightarrow S_i$ a "guarded command") amounts to the execution of an arbitrary S_i for which G_i holds. If $\neg(G_1 \vee \ldots \vee G_n)$ holds, the execution of the command is suspended until $(G_1 \vee \ldots \vee G_n)$ holds.

The execution of the ***repetition*** command $*[G_1 \rightarrow S_1 \| \ldots \| G_n \rightarrow S_n]$, where G_1 through G_n are boolean expressions, and S_1 through S_n are program parts, amounts to repeatedly selecting an arbitrary S_i for which G_i holds and executing S_i. If $\neg(G_1 \vee \ldots \vee G_n)$ holds, the repetition terminates.

For communication actions X and Y, "$X \bullet Y$" stands for the coincident execution of X and Y, i.e., the completions of the two actions coincide.

$[G]$ where G is a boolean expression, stands for $[G \rightarrow \textbf{skip}]$, and thus for "wait until G holds."

(Hence, "$[G]$; S" and $[G \rightarrow S]$ are equivalent.)

$*[S]$ stands for $*[\textbf{true} \rightarrow S]$, and thus for "repeat S forever."

From (iii) and (iv), the operational description of the statement $*[[G_1 \rightarrow S_1 \| \ldots \| G_n \rightarrow S_n]]$ is "repeat forever: wait until some G_i holds; execute an S_i for which G_i holds."

Communication commands: Let two processes, ***p1*** and ***p2***, share a channel with port X in ***p1*** and port Y in ***p2***. (In the processes of Figure 3, the same name is used for all the ports of the same channel.) If the channel is used only for synchronization between the processes, the name of the port is sufficient to identify a commnication on this port. If the communication is used for input and output of messages, the CSP notation is used: $X!u$ outputs message u, and $X?v$ inputs message v.

At any time, the number of completed X-actions in ***p1*** equals the number of completed Y-actions in ***p2***. In other words, the completion of the nth X-action "coincides" with the completion of the n-th Y-action. If, for example, ***p1*** reaches the nth X-action before ***p2*** reaches the nth Y-action, the completion of X is suspended until ***p2*** reaches Y. The X-action is then said to be ***pending***. When, thereafter, ***p2*** reaches Y, both X and Y are completed. It is possible (and even advantageous) to define communication actions as coincident and yet implement the actions in completely asynchronous ways. For an explanation, see [8].

Probe: Since we need a mechanism to select a set of pending communication actions for execution, we provide a general boolean command on ports, called the ***probe***. In process $p1$, the probe command $\overline{X}$ has the same value as the predicate "Y is pending in $p2$."

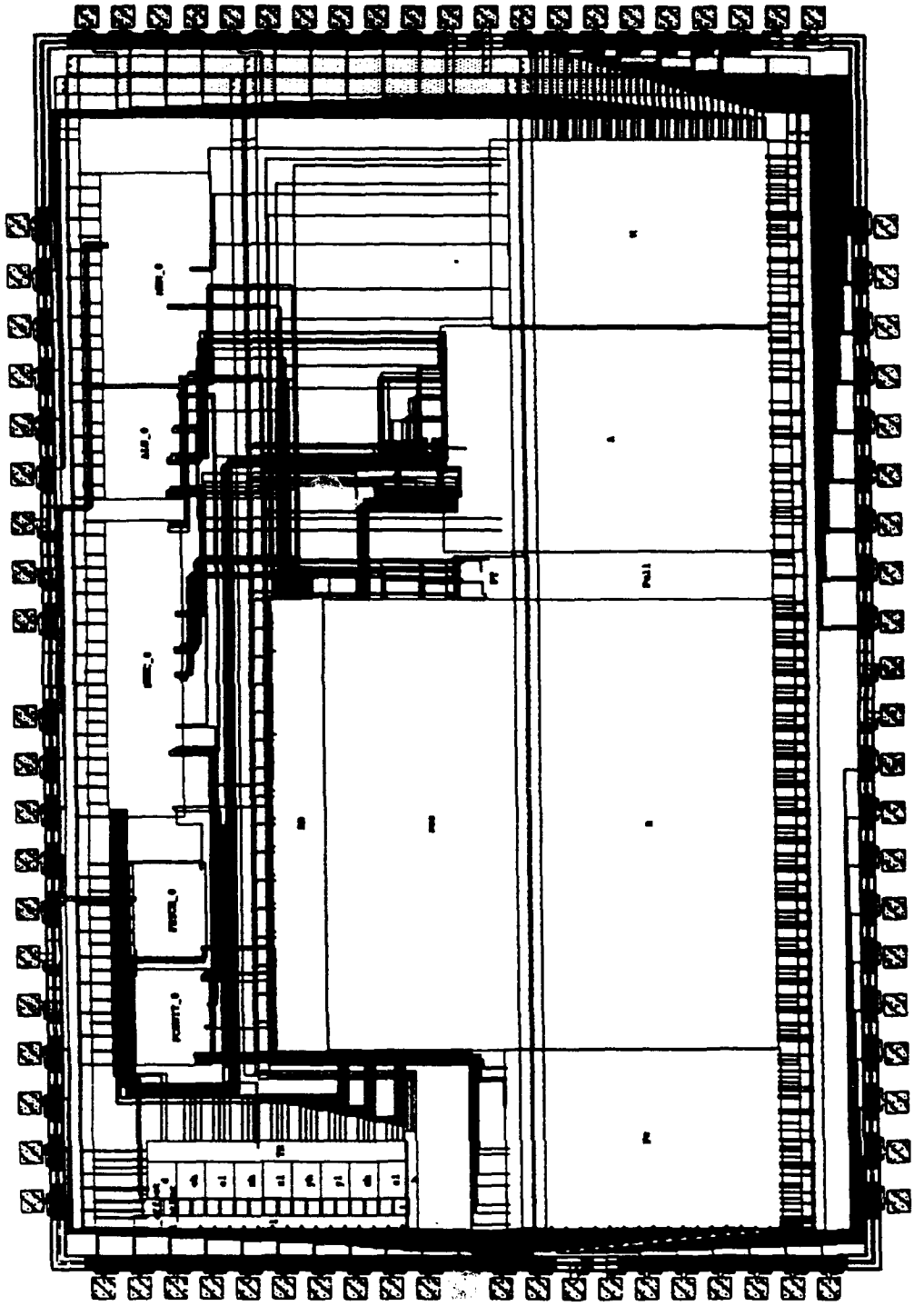
Appendix 2: Instruction Set

ALU	op rx ry rz	rz,f := rx op ry
MEM	op rx ry rz	rz := mem[rx+ry] (for load) mem[rx+ry] := rz (for store)
MEMOFF	op ao ry rz offset	rz := mem[ry + offset] (for load) mem[ry + offset] := rz (for store) rz := ry + offset (for load address)
BRANCH	op ao — cc offset	if cond(f,cc) then pc := pc + offset
JUMP	op ao ry —	pc := ry
STPC	op ao — rz	rz := pc

Table 1: Instruction Types

inst	n_3 $b_{15}b_{14}b_{13}b_{12}$	n_2 $b_{11}b_{10}b_9b_8$	n_1 $b_7b_6b_5b_4$	n_0 $b_3b_2b_1b_0$
alu	0011	rx	ry	rz
	0100	rx	ry	rz
	⋮			
	1111	rx	ry	rz
ld	0010	rx	ry	rz
st	0001	rx	ry	rz
ldx	0000	0000	ry	rz
stx	0000	0001	ry	rz
lda	0000	0010	ry	rz
brc	0000	0011	—	cc
jmp	0000	0100	ry	—
stpc	0000	0101	—	rz

Table 2: Opcode Assignments



Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm

Wen-King Su and Charles L. Seitz
Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-22

1. Introduction

We have been using variants of the Chandy-Misra-Bryant (CMB) distributed discrete-event simulation algorithm [1,2,3] since 1986 for a variety of simulation tasks [4]. The simulation programs run on multicomputers [5] (message-passing concurrent computers), such as the Cosmic Cube, Intel iPSC, and Ametek Series 2010. The excellent performance of these simulators led us to investigate a family of variants of the basic CMB algorithm, including lazy message-sending, demand-driven operation with backward demand messages, and adaptive adjustment of the parameters that control the laziness.

These studies were also motivated by our interest in scheduling strategies for reactive (message-driven) multiprocess programs [5,6,7], which are semantically similar to discrete-event (event-driven) simulators. The simulator itself is implemented in the reactive programming environment that we have developed for multicomputers: the Cosmic Environment and the Reactive Kernel [8].

We performed the studies reported here using logic networks. Logic simulation is expected to stress a distributed simulator, and is itself of practical interest. It is easy to construct examples of logic networks with a diversity of behaviors and structural difficulties, such as large fan-in and fan-out. Low-level logic elements such as logic gates exhibit responses in which an input event may or may not influence the outputs, depending on the internal state of the element and on the states of other inputs; yet, they require very little computation to simulate their behavior. Thus, the performance results shown later in this paper involve practically no computation other than the distributed simulation itself.

This paper is a brief and preliminary report of the simulation algorithms and performance results. A more definitive report will be found in the first author's forthcoming PhD thesis.

The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

2. The CMB Simulation Framework

As usual, the system to be simulated is modeled as a set of communicating elements. A CMB simulator can be implemented by coding the behavior of elements in processes that communicate by messages. A message conveys both a time interval and any events within this interval. A process reacts to the receipt of an input message by updating its internal state, and, if outputs can be advanced in time, by sending messages to connected processes. These messages may include *null messages* that convey no events (changes in the state information), but serve only to advance the simulation time.

It is easy to show that such a simulator is correct [3], in the sense that it computes a possible behavior of the system being simulated. A sufficient condition for freedom from deadlock in this eager message-sending mode is that there is a positive delay in every circuit in the graph of element vertices and communication arcs. Intuitively, it is the delay of the elements being simulated that permits the element simulators to compute the outputs over an interval that is later than the time of the inputs, so that time advances. Simulation time is determined locally, and may get as far out of step at different elements as their causal relationships permit.

This conservative (also known as pessimistic) type of simulator is a concurrent program that exploits the concurrency inherent in the system being simulated. In practice, just as with other concurrent programs, if the number of concurrently runnable processes substantially exceeds the number of processors, one can achieve high utilization of concurrent resources. The speculative (also known as optimistic) type of simulator attempts to exploit additional concurrency by computing beyond the interval during which inputs are defined, at the risk of having to roll back if the speculations prove incorrect. Such approaches are attractive for simulating systems whose inherent concurrency is insufficient to keep concurrent resources busy, and in which speculations can be made with high confidence. Our studies have concentrated on conservative variants of the CMB algorithm.

The design of distributed simulation programs is also influenced by a characteristic of the element simulators. In practice, an element simulator may or may not take as long to process a null message as an event-containing message. For the simulation of some systems, the processing of an event-containing message might involve a lengthy simulation of a physical process, whereas the processing of a null message might be very fast. Such simulations do not seriously stress the distributed-simulation aspect of the computation. However, for the simulation of systems of extremely simple elements, such as logic gates, the time required to compute the output of the gate is so small that it is comparable to the time required to process a null message.

Due to our interest in understanding the limits of event-driven distributed simulation, and the implications for scheduling strategies for message-driven multiprocess programs, our studies have concentrated on the case in which the time required to process null messages is comparable to the time required to process event-containing messages. It is straightforward to extrapolate the performance results for this difficult case to situations in which null-message processing is relatively fast.

The principal trouble with naive implementations of conservative CMB distributed simulation programs in any situation in which processing null messages is as costly as processing event-containing messages is that the volume of null messages may greatly exceed the number of event-containing messages. This difficultly is most evident when simulating systems with many short-delay circuits that have relatively low levels of activity.

In distributing the simulation, we seek to reduce the time required to complete the computation; however, we have an immediate problem if the element simulators must perform many more message-processing operations in the distributed simulation than they would perform event-processing operations in a sequential simulation. The centralized regulation of the advance of time achieved through the ordered event list maintained by sequential simulation programs allows these simulators to invoke element routines only once for each input event. The null messages inflate not only the volume of messages the system must handle, but also the computational load. Thus, if we are going to compete with the best sequential simulators, we must reduce the volume of null messages.

3. Indefinite Lazy Message Sending

To reduce the volume of messages, we use various strategies to defer sending outputs in the hope that the information can be packed into fewer messages. For example, one of the most obvious schemes is to defer sending null messages, so that a series of null messages and an event-containing message can be combined to form a single message that spans a longer interval. Since output events are often triggered only by input events, deferring the delivery of preceeding null messages is less likely to hamper the progress of the destination element than deferring the delivery of event-containing messages.

The first problem that must be addressed in employing such strategies is deadlock. When element simulators defer sending output messages, they may cyclically deny themselves input messages, leading to deadlock. All of our simulators have employed a technique of *indefinite lazy message sending* to permit arbitrary strategies for deferring message sending while still avoiding deadlock. The following is an idealized inner loop of the simulator, shown in the C programming language:

```
while(1)
  if (p = xrecv())
     simulate_and_optionally_send_messages(p);
  else
     take_other_action();
```

The function `xrecv` returns a pointer, p, that points to a message for the simulation process if a message has been received. The simulator then dispatches to the appropriate element simulator, and may either send or queue the outputs that the element simulator produces. If there is no message in the node's receive queue, the pointer returned is a NULL (0) pointer. In this case, the simulator takes other

action to break any possible deadlock. For a source-driven simulator, it selects a queued output to send as a message. For a demand-driven simulator, it selects a blocked element, and sends a *demand* message to its predecessor to request that queued outputs be sent. A deadlock in deferring messages cannot occur without "starving" a node of messages. When this situation is detected by `xrecv` returning a NULL pointer, the resulting action breaks the potential deadlock.

Within this indefinite lazy message-sending framework, we can experiment with *any* scheme for deferring and combining messages without concern for deadlock. A message is free to carry any number of events, and an element is free to defer message sending on any basis.

4. Variant Algorithms

We have experimented with many CMB variants; in the interests of comprehension, we will describe the operation and report the performance of six that are representative of the range of possibilities that we have studied:

A *Eager message sending:* This basic form of CMB serves as a baseline for comparison against the variants.

B *Eager events, lazy null messages:* Null outputs are queued. Event outputs, combined with any queued null outputs, are sent immediately. When `xrecv` returns a NULL pointer, the null output that extends to the earliest time is sent as a null message.

C *Indefinite-lazy, single-event:* All output from element simulators is queued. The output queues may contain multiple events. Messages are sent only when `xrecv` returns a NULL pointer. The output queue that extends to the earliest time is selected to generate a message up to the first event, if any, or a null message to the end of the interval.

D *Indefinite-lazy, multiple-event:* This scheme is a slight variation on *C*, motivated by characteristics of multicomputer message systems that make it economical to pack multiple events into fewer messages. All output from element simulators is queued. The output queues may contain multiple events. When `xrecv` returns a NULL pointer, the output queue that extends to the earliest time is selected to generate a message up to the *last* queued event, if any, or a null message to the end of the interval. However, to allow a direct comparison with sequential simulators, events are processed singly.

E *Demand-driven:* Although we usually think of simulation as source driven from inputs, one can equally well organize the simulation as demand driven from outputs. In the pure demand-driven form, all output from element simulators is queued. When `xsend` returns a NULL pointer, the input that lags furthest behind selects the destination for a demand message. Upon receipt of a demand message, if the output queue is not empty, the simulator sends all the information in the output queue; if the output queue is empty, the simulator generates another demand message to the source of lagging input to this element.

F Demand-driven, adaptive: Demand messages single out critical paths in a simulation. In an adaptive form of demand-driven simulation, a threshold is associated with each communication path. Outputs of element simulators are queued only up to the threshold; when the threshold is exceeded, the contents of the queue are sent as a message. Demand messages operate as in *E*, but also cause the threshold to be decreased. In the cases shown below, the threshold is halved. The simulator is accordingly able to adapt itself to the characteristics of the system being simulated.

Although these variants are described here in terms of message passing, the same variants also appear as different scheduling strategies in shared-memory implementations.

5. Experimental Method

In common with other highly evolved message-passing programs, the simulator is implemented with one simulation process per multicomputer node (or, in the Cosmic Environment, with one simulation process per host computer or per processor in a multiprocessor).

Basis of comparison: Although execution time is one of the most natural bases of comparison between any two programs that perform the same function, and is used below to illustrate the performance of our distributed simulators on different commercial multicomputers, execution time on these concurrent computers depends both on the algorithm and on the characteristics of the particular computer. When we wish to isolate the characteristics of the algorithm from those of the computer, the instrumented simulator operates as a simulator within a simulator. Execution time is then measured in a unit called a *sweep* [5, 6], which corresponds here to a fixed time required to call an element once. The time required for other operations, such as sending a message, can be set to a particular number of sweeps. Normally, a message sent by one node in one sweep is available in the destination node at the next sweep. However, to test the sensitivity of the algorithms to message latency, we can also set the latency to larger values.

Instrumentation: The simulator is a reactive program written in C, and is instrumented to function in two operational modes. In the *sweep mode*, a multicomputer-emulation program runs a simulation of a multicomputer; this in turn runs the reactive simulators. Time is measured in sweep units; on each sweep, each node is allowed to make one element call. In the *real mode*, the simulator runs directly on the multicomputer. There is one copy of the simulator process in each node, and each simulator process runs a subset of the elements as embedded reactive processes. Each node runs at its own pace, and execution time is measured with UNIX's real-time clock.

6. Experimental Results

Performance measurements have been made on a variety of logic networks, including those that are representative of networks found in computers and VLSI chips, and

those that are designed specifically to test or to stress the simulator. Six different network types, each in several sizes up to 4000 logic gates, have been the principal vehicles for these experiments. A larger variation in performance is observed among networks with different characteristics than between algorithm variants.

Multiplier example: The parallel multiplier is a good example of an ordinary logic network. The 14×14 multiplier used in several experiments employs 1376 logic gates to generate the 28-bit product of two 14-bit binary inputs. The multiplier network contains only limited concurrency, and does not contain tight circuits that give the simulator artificial performance boosts or troubles, depending on element distribution. It also contains moderately high fan-out in the multiplier and multiplicand lines; this puts pressure on the message system. In all fairness, the distributed simulation of this multiplier network is not expected to do too badly nor too well.

For the simulation, the most-significant bit of the product is connected back to the multiplier input via an inverting delay. The delay is such that the multiplier reaches a stable state before the multiplier input changes. The multiplicand input is set to a value that causes the circuit to oscillate. A trace of the product outputs shows that the simulator and the circuit are running correctly.

Measurements in the sweep mode: The plot in Figure 1 portrays in a log-log format the sweep count in the sweep mode versus the number of nodes, N, for the simulation of the 14×14 multiplier network under all six CMB variants. It is not useful to continue the plot beyond 2^{11} nodes, since at this point there are as many nodes as simulated gates. The placement of elements in nodes for these trials is balanced but random.

Each horizontal division represents a factor of two in resources; each vertical division represents a factor of two in sweep count or time. We have found this format (*cf* [5]) for portraying the performance of concurrent programs to be more useful than "speedup" graphs, for two reasons. First, we can observe the factor by which the execution time is reduced as resources are increased over very wide ranges. Second, since the ordinate is a physical measure, time or sweep count, we can compare different algorithms directly. For example, in addition to the plots of the sweep counts of the CMB variants, the heavy horizontal line represents the number of sweeps a sequential simulator requires for this same simulation.

The first remarkable characteristic of these performance measurements is that they are so similar across this class of variant algorithms. Algorithms A, E, and F produce more messages than B, C, and D, but in this mode in which messages are free but element invocations are expensive, there is little difference between the variants. The performance under sweep-mode execution exposes the intrinsic characteristics of the algorithm, and is not related to such multicomputer characteristics as the relationship between node computing time and message latency.

The gross characteristics of these curves are similar to those of other concurrent programs [5], and are quite understandable and predictable.

We observe at $\log_2 N{=}0$ (1 node) that all of the CMB variants are somewhat inefficient in comparison with the sequential event-driven simulator. For this

multiplier example, the null messages inflate the number of element invocations by a factor of 2–5 times; this is consistent with the 1–2.5-octave increase in sweep count over that of the sequential simulator. The null messages also inflate the concurrency over that which is intrinsic to the system being simulated. We shall refer to this inflation in the number of element invocations as the *overhead* of distributing the simulation. If the time required to process a null message were smaller than the time required to process an event-containing message, the overhead would be reduced proportionately.

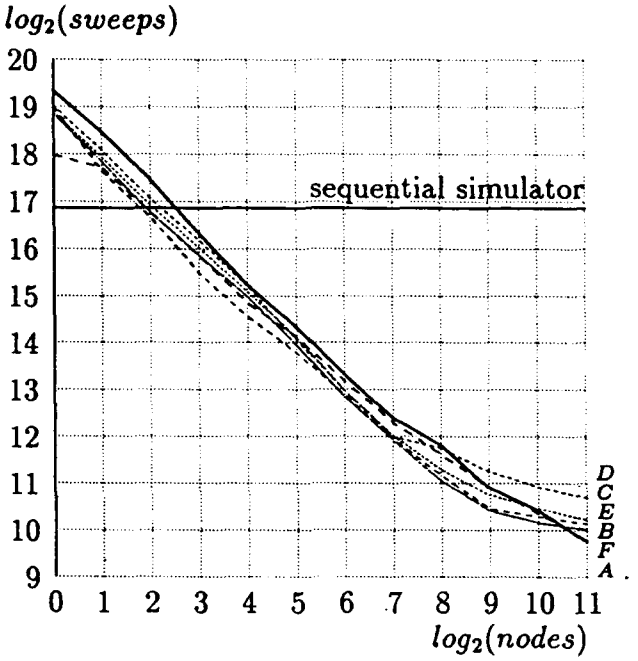


Fig 1: A 1376-gate multiplier, sweep mode

The performance is then divided roughly into two regimes, the first regime being one of near-linear speedup in N for the first 7–8 octaves, and the second regime being one of diminishing returns in N as the computing time approaches an asymptotic mimimum value. In the linear speedup regime, these simulators nearly halve the sweep count with each doubling of resources until limiting effects are reached. Load balance is assured by the weak law of large numbers when there are many elements per node. While each node has a sufficiently large pool of work, node utilization remains high. The simulators approach asymptotic minimal time as they exhaust the available concurrency in the system being simulated. The gradual "knee" of the curve originates from progressively less-effective statistical load balancing as the number of elements per node diminishes with larger N.

Additional statistics have been collected to measure other effects. For example, in the linear-speedup regime, when there are many logic elements per node, the simulators are quite insensitive to message latency. When there are few elements per node, the performance begins to deterioriate as message latency is increased. These

effects will be evident in the measurements performed on real multicomputers.

Measurements on real multicomputers: The results of simulating the same 1376-gate multiplier network on a 16-node iPSC/2 is shown in Figure 2, and on a 128-node iPSC/1 for variants B, C, and D is shown in Figure 3. The iPSC/2 is $\approx$6 times faster per node than the iPSC/1, so the time scales do not correspond. This simulation will not run on an iPSC/1 for $N < 4$ because the data and message queues for an increased number of logic elements per node will not fit in the node memory. Due to the same limitations of the iPSC/1 message system, neither the demand-driven nor the eager-message-sending simulation variants will run in most machine sizes. This choice of performance data is dictated by the desire to show performance results over the largest range of N possible with the machines that are currently operated by our research group. Results essentially identical to those shown in Figure 2 are also obtained on a 16-node Ametek Series 2010.

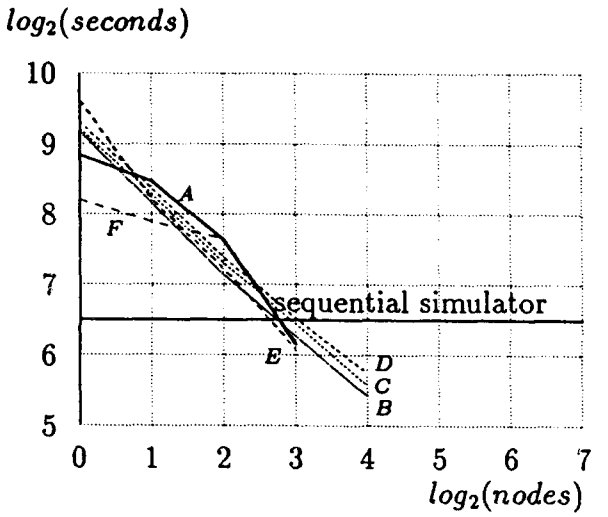


Fig 2: A 1376-gate multiplier for 40μs on an iPSC/2

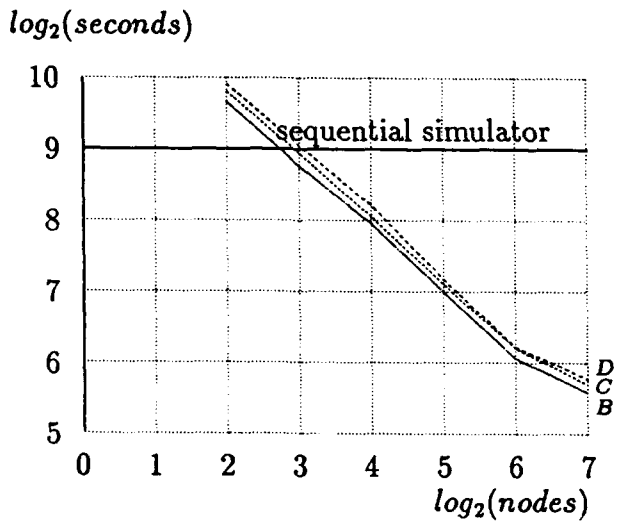


Fig 3: A 1376-gate multiplier for 40μs on an iPSC/1

The simulation of this network for $2^0 \leq N \leq 2^7$ is in the relatively uninteresting (but useful) linear-speedup regime, with some limiting effects starting to be seen in Figure 3 at $N=2^7$. The number of gates being simulated per node is sufficiently high to keep the node utilization high and the sensitivity to message latency low.

In order to exhibit the performance results in the more interesting (but less useful) diminishing-returns regime, we have scaled the network down to a 4-bit multiplier with 116 logic gates. The peformance on an Intel iPSC/2 up to 16 nodes is shown in Figure 4, and on an Intel iPSC/1 up to 128 nodes is shown in Figure 5. This network is small enough to exhibit interesting limiting effects as the simulation is increasingly distributed. The sublinear speedup is due to message latency in inter-node communications, increased null messages as the simulation is increasingly distributed, and load imbalance. The asymptotic time is limited by the message latency rather than by the available concurrency. In particular, Figure 5 shows that the asymptotic execution time of algorithm A, which is not very economical in its use of messages, is more than a factor of two worse than the asymptotic execution time of variants B, C, and D.

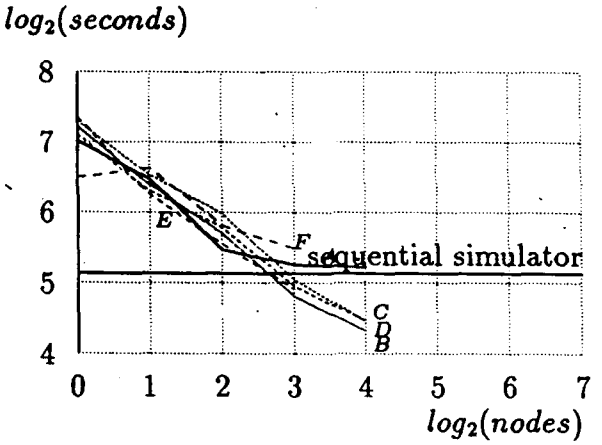


Fig 4: A 116-gate multiplier for 100μs on an iPSC/2

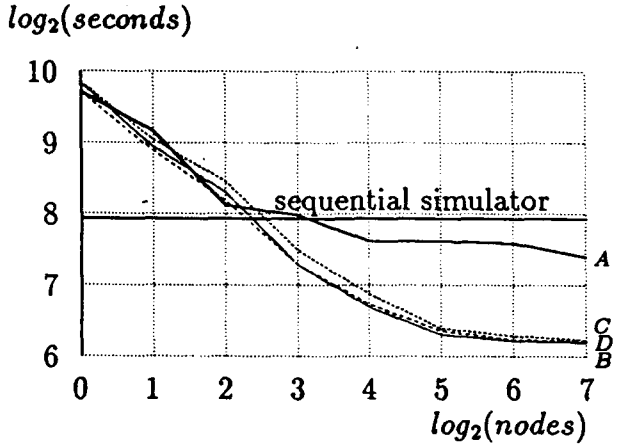


Fig 5: A 116-gate multiplier for 100μs on an iPSC/1

7. Hybrid CMB Variants

Although the CMB variants exhibit good speedup over wide ranges of N, speedup measures only the performance of the algorithm relative to less-distributed instances of itself. In comparison with the sequential simulator, the distributed simulators must pay the overhead of processing null messages. If the elements used in a simulation are such that the time required to process null messages is considerably less than the time to process event-containing messages, these conservative CMB variants will provide excellent performance and efficiency.

However, if the time required to process null messages is comparable to the time required to process event-containing messages, as it is for logic simulation, this overhead makes the CMB algorithm and its variants problematic for simulations on parallel computers in which N is small. What might be done to extend the CMB approach into this difficult small-N range?

A component of the overhead that cannot be eliminated within the CMB framework, in which elements are independent processes, is the null messages used to force progress in cycles of idling elements. However, we can take advantage of multiple elements sharing the same node by lumping members of low-latency, low-activity cycles, such as the gates that form a latch, into macro elements, and applying sequential simulation to them internally. The null-message-processing overhead for such cycles is eliminated at the cost of reduced concurrency for their members.

In this type of hybrid CMB variant simulator, all elements in each node are combined into one macro element, which is simulated internally with a conventional, ordered-event-list, sequential simulator. These sequential simulators are tied together externally with one of the CMB variant simulators. Since there is only one macro element per node, the hybrid variants are identical at N=1 to a sequential simulator. As N increases, however, more cycles are partitioned over multiple nodes, and each hybrid variant eventually converges with its corresponding CMB variant.

Measurements in sweep mode: Figure 6 shows the performance results for the CMB variants simulating a ring of 28 self-timed FIFO units. Each FIFO unit contains one FIFO-control cell and eight register cells, implemented with a total of 1067 logic gates. The FIFO ring is 50% full, holding 14 alternating 1- and 0-bytes. The overhead at N=1 is caused by the idling of the cross-coupled NAND latches in the registers and the FIFO controls. The CMB variants show a good speedup with increased N. Except for the initial overhead, the performance of all of the CMB variants is excellent.

Figure 7 shows the simulation results for the same circuit using the hybrid CMB variants with an element-distribution method that tends to place elements of each cycle in the same node.

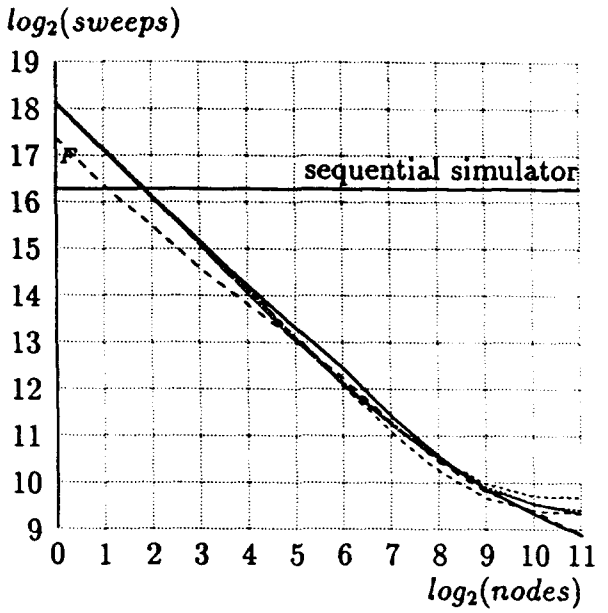


Fig 6: FIFO ring, non-hybrid simulator, emulation mode

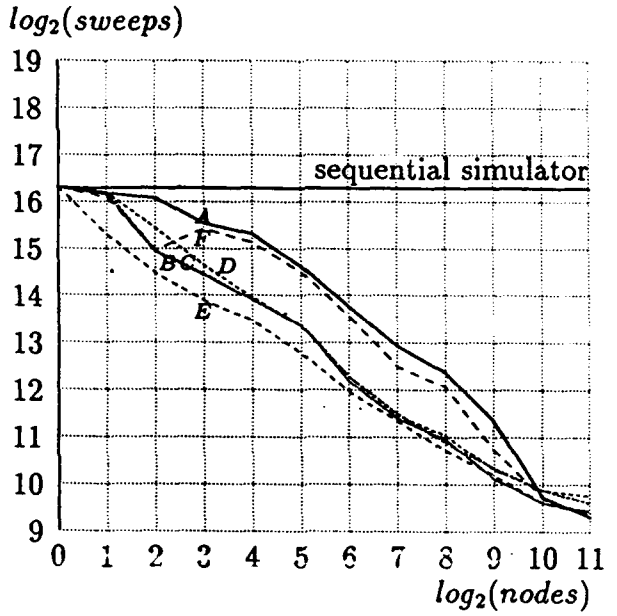


Fig 7: FIFO ring, hybrid simulator, emulation mode

Although the hybrid simulator exhibits a generally decreasing sweep count with increasing N, and extremely good small-N performance for the demand-driven variant E, less desirable behaviors have been observed for the hybrid variants. In particular, if the elements are not properly distributed, or cannot be properly distributed, the simulation time may increase starting at N=2 before starting to decrease. This effect is the result of cycles being broken and scattered over multiple nodes, so that it is the CMB rather than the sequential algorithm that dominates the execution time. Figure

8 illustrates the performance of the simulator for the same circuit used in Figures 6 and 7, but with random placement of the elements across the nodes.

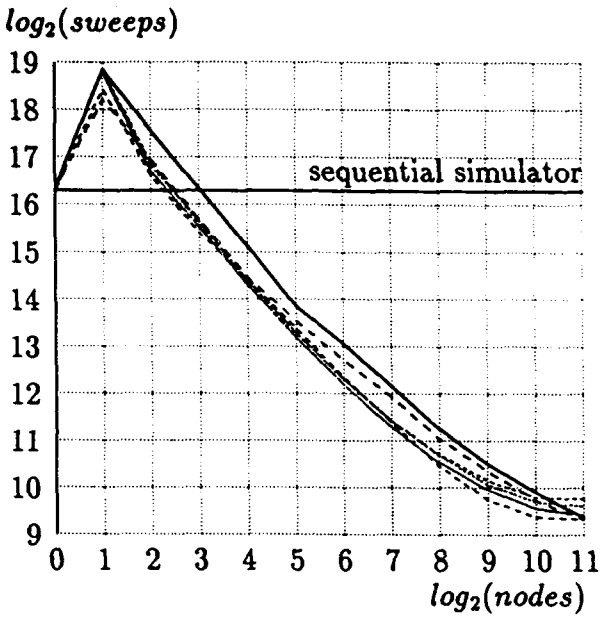


Fig 8: FIFO ring, hybrid simulator, randomized

Some programming short-cuts were used to produce these sweep-mode performance measures for the hybrid variants without implementing a regular sequential simulator; thus, we are not able to include corresponding performance graphs for real multicomputers. However, the instrumentation of the hybrid sweep-mode simulations, together with the performance parameters of second-generation multicomputers such as the Intel iPSC/2 and Ametek Series 2010, indicate that the performance on real multicomputers will be essentially similar to that in the sweep-mode. We are currently implementing distributed simulation programs and instrumentation to run the hybrid CMB variants on real multicomputers.

8. Conclusions

We selected logic simulation for these experiments because we wished to examine the limits of the applicability of the conservative CMB algorithm and its variants. Simulating the behavior of relatively simple elements that have a high degree of connectivity was expected to be a difficult case for distributed simulation. Indeed, the performance results presented here have been much more revealing of the capabilities and limitations of the distributed discrete-event simulation algorithms than earlier simulations that we performed of systems such as multicomputer message networks.

The reader should accordingly be cautious about drawing negative conclusions about the CMB framework from our comparisons of the performance of the CMB variants with the ordered-event-list sequential simulator. For objects of distributed simulation that are less demanding than logic simulation, such as systems in which

processing null messages is much faster than processing event-containing messages, the overhead is proportionately scaled down, and the following general conclusions remain valid:

1. Selected CMB varients exhibit excellent speedup over a wide range of N, limited eventually only by the concurrency of the system being simulated.
2. The CMB variants presented here, all based on the indefinite-lazy-message-sending framework, provide a useful improvement over the basic eager-message-sending CMB algorithm.
3. The hybrid CMB variants offer promise of efficient distributed simulation on small-N concurrent computers.

In some respects, the CMB and sequential algorithms make poor comparison subjects because these two algorithms represent relatively orthogonal optimizations in the basic task of simulation. While the execution time of the sequential simulator is sensitive only to the activity level of the circuit, the execution time for the fully distributed CMB algorithm is sensitive only to the structure of the circuit. In the FIFO-ring example, we can use more data bytes, fewer data bytes, or a different set of data bytes, and shift the sequential simulator's execution time proportionately without significantly changing the CMB variants' curves. Similarly, we can shift the CMB variants' curves without affecting the execution time of the sequential algorithm by varying the delay of the gates in the latches.

The hybrid CMB variants attempt to combine the best aspects of the sequential and CMB algorithms by allowing the sequential simulator to dominate when N is small, and the CMB variants to dominate when N is large. This approach may or may not produce a favorable result, depending on whether the elements can be properly distributed. More research needs to be done in the area of element distribution and its effect on the hybrid variants.

9. Acknowledgment

We very much appreciate the constructive suggestions, ideas, and encouragement that we have received from K. Mani Chandy.

10. References

[1] K. Mani Chandy and Jayadev Misra, "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations," *CACM* 24(4), pp 198–205, April 1981.

[2] Randal E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[3] Jayadev Misra, "Distributed Discrete-Event Simulation," *Computing Surveys* 18(1), pp 39–65, March 1986.

[4] "Submicron Systems Architecture," Semiannual reports to DARPA, Caltech Computer Science Technical Reports [5220:TR:86] and [5235:TR:86], 1986.

[5] William C. Athas and Charles L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer* 21(8), pp 9–24, August 1988.

[6] William C. Athas, "Fine Grain Concurrent Computation," Caltech Computer Science Technical Report (PhD thesis) [5242:TR:87], May 1987.

[7] William J. Dally, *A VLSI Architecture for Concurrent Data Structures,* Kluwer Academic Publishers, 1987.

[8] Charles L. Seitz, Jakov Seizovic, and Wen-King Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming," Caltech-CS-TR-88-1, January 1988.

The Essence of Distributed Snapshots

K. Mani Chandy*
California Institute of Technology

6 March 1989
Caltech-CS-TR-89-5

1 Introduction

A distributed system has no global clock, and it is the absence of a global clock that makes for several interesting problems, one of which is obviously important, but apparently trivial: 'Record the state of the system.' Recording the state of distributed system is called 'taking a global snapshot' after [2]. If there were a clock, taking global snapshots would be straightforward: Each process records its state or at some predetermined time, and the collection of recorded process states is used to construct a system state.

Global snapshots are useful in a variety of situations [2,3,6]. The goal of this paper is to identify the essential properties of global snapshots so as to simplify proofs of global snapshot algorithms and to aid in the design of new algorithms.

2 A Distributed System

2.1 Standard Definitions

We shall first define a distributed system as in [8].

*Supported in part by DARPA-6202, monitored by ONR N00014-87-K-0745

A prefix of a sequence z is an initial subsequence of z. A prefix-closed set of sequences is a set such that every prefix of a sequence in the set is also in the set.

A system is a set of *components*. A component is a set of *events* and a prefix-closed set of sequences of its events called its set of *computations*.

A *projection* of a sequence v on a component is the sequence obtained from v by deleting all events in v that are not events of the component.

A *system computation* is a sequence v of events of components of the system such that the projection of v on each component of the system is a computation of that component.

Let $w.p$ be a computation of component p, all p. Let P be a set of components. An *interleaving* of a set of component computations $\{w.p \,|\, p \in P\}$ is a sequence, v, of events of components in P, such that the projection of v on p is $w.p$, all $p \in P$.

We use (y, z) for the catenation of sequences y and z.

2.2 Processes and Channels

A component of a distributed system is either a *process* or a *channel*. Distinct processes have disjoint sets of events, and distinct channels have disjoint sets of events.

A channel is *used by* exactly two processes. The events of a channel are events of the processes that use the channel. We shall restrict attention to channels that satisfy the following monotonicity condition.

Let c be a channel used by processes q and r. Let u, v be computations of c, where $u.r = v.r$, and $u.q$ is a prefix of $v.q$. Let e be an event on r.

A Monotonicity Property If (u, e) is a computation of c, then (v, e) is also a computation of c.

Explanantion The monotonicity condition implies that the execution of events on one process cannot inhibit the execution of an event on another process. If a channel c is used by processes q and r, and there is a computation of c in which e is executed on process r after q and r have executed computations a and b respectively, then there is a computation

of c in which e is executed after r has executed b, and q has executed *an arbitrary sequence of events following a.*

Example: Bounded First-In-First-Out Buffers Consider a first-in-first-out buffer, with a capacity of N messages ($N > 0$), shared by a single producer process and a single consumer process. Such a buffer is a channel that has the monotonicity property, as shown by the following informal argument.

The producer can append any message to the buffer if the buffer is not full. The consumer can receive a message m from the buffer if the buffer is not empty and m is the message at the head of the buffer queue. If the producer can produce a message after it has produced i messages and the consumer has consumed j messages, then the producer can produce a message after it has produced i messages, and the consumer has consumed more than j messages. Therefore, the monotonicity property holds with r as the producer and q as the consumer.

By a similar argument, additional production does not prevent the consumer from receiving the message at the head of the buffer; the monotonicity property also holds with r as the consumer and q as the producer. Therefore, the channel has the monotonicity property.

Example: Stacks Next consider a channel which is a stack. Let m be the message at the top of the stack, if the stack is not empty. The consumer can execute the event: Pop the stack and consume m. The producer can execute an event: Push a message m' on top of the stack. Such a buffer does not have the monotonicity property because an event of the producer — push m' on the stack — where $m \neq m'$, can prevent the consumer from executing the event: Pop the stack and receive message m. Therefore, an event on one process can inhibit the execution of an event on the other.

Note: Symmetry of Processes One way of defining channels is in terms of causality: one of the processes sends a message on the channel, and the other receives the message, thus there is a causal relationship between the sending and the receiving of the message. The definition of channels used in this paper is symmetric with respect to both processes; the defini-

tion does not employ the concept of causality. Monotonicity appears to be an important property of channels of distributed systems.

3 The Problem

Restrict attention to one given system. Let z be a finite computation of the system. For ease of exposition, assume that all events in z are distinct. (If events are repeated in z, then number the events, so that the pair — event-name, number — is distinct.) Let $z.p$ be the projection of z on a process p. Let $x.p$ be any prefix of $z.p$. Let S be the set of process computations $\{x.p | p \text{ is a process } \}$.

Set, S, is defined to be a global snapshot in z if and only if there exists a system computation y where:

1. y is an interleaving of the set of process computations $z.p$, and
2. every event in S occurs in y before every event that is not in S.

The problem is to determine simple necessary and sufficient conditions for S to be a global snapshot in z.

Motivation for the Problem Set S is a global snapshot in z if and only if there is a system computation that first takes the system to a state in which each process p has executed $x.p$, and then to the state in which each process p has executed $z.p$. Informally, S is a global snapshot in z if and only if it *could have* happened that all events in S were executed before all events that are not in S. If S is a global snapshot in z, then properties about S can be used to deduce properties about the state of the system after z is executed. Therefore, it is helpful to determine simple conditions that guarantee that S is a global snapshot.

4 A Solution

The obvious algorithm to determine if S is a global snapshot in z is as follows: Since z is finite, enumerate all interleavings of $z.p$, and determine if there is one with the desired properties. This approach is computationally

intractable if the number of processes is large. Next, we present a theorem that helps us to design tractable solutions.

4.1 Compatible Computations

Let c be a channel. Let c be used by processes q and r. Let u and v be computations of q and r respectively. Process computations u and v are defined to be *compatible with respect to* c if and only if there exists an interleaving w of u and v such that the projection of w on c is a computation of c.

Informally, u and v are compatible with respect to c if and only if process computations u and v could have occurred in a computation of a system consisting of only the two processes q and r, and the single channel c.

Example: Bounded First-In-First-Out Buffers Let c be a channel that is a first-in-first-out buffer with a capacity of N where $N > 0$, and where the buffer is initially empty. Let u and v be computations of the processes that send and receive (respectively) on the channel. Then u and v are compatible with respect to c if and only if the sequence of messages received along c in v is a prefix of the sequence of messages sent along c in u, and the number of messages sent along c in u exceeds the number of messages received along c in v by at most N.

Let z and $x.p$ be as in the problem definition, i.e., z is a system computation and $x.p$ is a prefix of $z.p$. Let the producer and consumer for c be q and r respectively. Since z is a system computation, the sequence of receives along c in $z.r$ is a prefix of the sequence of sends along c in $z.q$. Therefore, the sequence of receives along c in $x.r$ is a prefix of the sequence of sends along c in $x.q$ if and only if the number of receives along c in $x.r$ is at most the number of sends along c in $x.q$. Therefore $x.q$ and $x.r$ are compatible with respect to c if and only if

$$0 \leq (nsends - nreceives) \leq N$$

where *nsends* and *nreceives* are the numbers of sends and receives along channel c in $x.q$ and $x.r$ respectively.

4.2 The Snapshot Theorem and its Applications

We shall first give the theorem, discuss its consequences, and then prove it. Let z, $z.p$, $x.p$ and S be as given earlier.

Theorem Set, S, is a global snapshot in z if and only if, for each channel, c:
$x.q$ and $x.r$ are compatible with respect to c, where q and r are the processes that use r.

Applications of the Theorem The proof that S is a global snapshot of an arbitrary system reduces to a proof of compatibility of a pair of process computations for each channel. Let us use this fact in developing algorithms for a couple of problems. The following discussion is very brief and informal, because our goal is only to demonstrate the use of the theorem, rather than to give a thorough exposition of the algorithms.

The Snapshot Algorithm We shall develop the algorithm in [2]. Consider a system in which channels are first-in-first-out and the capacity of a channel is arbitrarily large. Initially all channels are empty. We wish to develop a distributed algorithm to record the global state of the system.

Consider a channel c used by processes q and r, where q sends along c, and r receives along c, and initially c is empty. As discussed earlier, process computations $x.q$ and $x.r$ are compatible with respect to c if and only if the number of receives along c in $x.r$ is at most the number of sends along c in $x.q$. Therefore the problem of algorithm design reduces to this: Guarantee that the number of receives before the receiver records its state is at most the number of sends before the sender records its state, and also guarantee that every process records its state eventually.

One way of doing this is as follows: After a process records its state, it sends a special message called a *marker* along each of its outgoing edges. On receiving a marker a process records its state if it has not done so. At least one process (called the initiator) records its state in finite time; if there is a path (a sequence of channels) from the initiator to all other processes then every process records its state in finite time of the intiator.

Logical Time Consider the same system as in the previous paragraph. Let z be a computation of the system. We are required to give each event in z a unique number, called its logical time, such that the set of events with logical times less than n corresponds to a global snapshot in z, for all n. Let $x.p$ be the prefix of $z.p$ consisting of all events with logical times less than n; we require that the set S (defined as before as $\{x.p\}$) be a global snapshot.

As in the previous problem, the problem of algorithm design reduces to this: Guarantee that for each channel c, the number of messages received along c in $x.r$ is at most the number of messages sent along c in $x.q$, where q and r are the processes that send and receive along c, respectively. This is equivalent to: guarantee that logical times of events are such that the event of receiving a message has a higher logical time than the event of sending that message. One way of doing so is in [7]: put a time-stamp t on each message where t is the logical time of the event that sends the message, and give the event that receives a message a logical time that is greater than the time-stamp of the message.

(The goal for logical time in the seminal paper [7] is different from that given here, because it is based on the concept of causality. Our goal here is limited: to demonstrate a use of the snapshot theorem.)

4.3 Proof of The Snapshot Theorem

Snapshot Theorem Let z be a finite computation of the system. Let $x.p$ be a prefix of $z.p$, all p. Let S be the set of process computations $\{x.p | p$ is a process $\}$. Set S is a global snapshot in z if and only if, for each channel c, computations $x.q$ and $x.r$ are compatible with respect to c, where q and r are the processes that use c.

Proof If $x.q$ and $x.r$ are incompatible with respect to c, then there is no interleaving of $x.q$ and $x.r$ that is a computation of c, and hence S is not a global snapshot. Next, we prove that if for each c, $x.q$ and $x.r$ are compatible with respect to c, where q, r use c, then S is a global snapshot. The proof given here is a generalization of the proofs given in [2,5] which are limited to unbounded first-in-first-out channels.

Define sequence y as follows: y is the permutation of z where all events in S occur before all events that are not in S, and apart from this change, the order of events in z is retained in y. We shall prove that S is a global snapshot by proving that y is a system computation.

Let w be a permutation of z. We shall give an algorithm which starts with $w = z$ and that ends with $w = y$, and where the algorithm maintains the invariant: w is a system computation.

The Algorithm Initially $w = z$. While w contains a pair of adjacent elements d and e, where d occurs before e, and d is not in S, and e is in S: interchange the positions of d and e in w.

Proof of Termination We prove that the algorithm terminates in a finite number of steps by using the metric: the number of pairs (f, g), where event f occurs earlier than event g in w, and f is not in S, and g is in S. The algorithm terminates if and only if the metric is zero, in which case $w = y$.

The metric has a finite value initially, and every step decreases it; hence the algorithm terminates in a finite number of steps.

Proof of the Invariant We prove the stronger invariant that $w.p = z.p$, all processes p, and $w.c$ is a computation of c, all channels c, where $w.d$ and $z.d$ are the projections of w and z, respectively, on component d. The invariant holds initially, because $w = z$. Let w' be the same as w except that d and e are interchanged. Our proof obligation is to show that w' satisfies the invariant if w satisfies it.

Since $x.q$ is a prefix of $z.q$ and since $w.q = z.q$, it follows that $x.q$ is a prefix of $w.q$. Therefore, if two adjacent events in w are on the same process, q, and the first of the two events is not in $x.q$, then the second is not in $x.q$ either. Since d is not in S and e is in S, it follows that d and e cannot be on the same process. Let d be on process q and let e be on process r, where $r \neq q$.

Since d and e are on different processes, $w'.p = w.p$, all p, and therefore $w'.p = z.p$.

If d and e are on different channels, then the projections of w and w' on each channel are identical, and hence the invariant holds for w'. Therefore, we need only consider the case where d and e are on the same channel; let this channel be c. Our only remaining proof obligation is to show that $w'.c$ is a computation of c.

Let t be the prefix of w preceding d in w. Then (t, d, e) is a prefix of w, and (t, e, d) is a prefix of w'.

Since $x.q$ and $x.r$ are compatible with respect to c, there exists an interleaving h of $x.q$ and $x.r$ such that the projection of h on c is a computation of c. Event e is in $x.r$, and therefore is in h. Define u as the prefix of h preceding e. Therefore, (u, e) is a prefix of h, and hence it is a computation of c. Since both $u.r$ and $t.r$ are the prefixes of $x.r$ that precede e, it follows that $u.r = t.r$. Since d is not in $x.q$, it follows that $x.q$ is a prefix of $t.q$. Since $u.q$ is a prefix of $x.q$, it follows from the transitivity of the prefix relation that $u.q$ is a prefix of $t.q$. From the monotonicity property, the projection of (t, e) on c is a computation of c.

Applying the monotonicity property again, the projection of (t, e, d) on c is a computation of c, since the projections of (t, d) and (t, e) on c are computations of c.

Let m be the length of the sequence (t, e, d). We shall prove by induction on k, that for $k \geq m$: $g'.c$ is a computation of c where g' is the prefix of w' of length k.

Base Case: $k = m$. This case has already been proved.

Induction Step: Let f be the $(k+1)$-th event in w. Our proof obligation is to show that the projection of (g', f) on c is a computation of c. Let g be the prefix of w of length k. The projection of (g, f) on c is a computation of c because (g, f) is a prefix of w. From the induction hypothesis, the projection of g' on c is a computation of c. For $k \geq m$: $g.q = g'.q$ and $g.r = g'.r$. If f is on c, the from the monotonicity property of c, the projection of (g', f) on c is a computation of c. If f is not on c, then the projection of (g', f) on c is the same as the projection of g' on c, and the result follows.

5 Partial Snapshots

There are some problems in which a snapshot of some subset of processes and channels is useful, and a global snapshot of all processes and channels is *not* necessary. We define a *partial* snapshot of a set of processes, Q, in a manner analogous to the definition of a global snapshot. Let z be a system computation. Let $x.p$ be a prefix of $z.p$. Let S be the set of process computations $\{x.q | q \in Q\}$. Set S is defined to be a partial snapshot in z if and only if there exists a system computation y where:

1. y is an interleaving of the set of process computations $z.p$, all processes p, and

2. for each process q in Q, the events in $x.q$ appear in y before the events of q that are not in $x.q$.

A partial snapshot is a global snapshot if Q is the set of all processes.

Next, we shall define a class of problems for which partial snapshots are helpful.

5.1 Termination Problems

Let w be a system computation. Set, Q, is defined to have terminated after w if and only if,

1. for all events e, and all processes q in Q, if $(w.q, e)$ is a computation of q, then e is an event on a channel between q and a process in Q, and

2. for all channels c between processes in Q, there is no event e such that $(w.c, e)$ is a computation of c.

Informally, the first condition says that after a process q has executed $w.q$ it can only execute events on channels connecting it to other processes in Q. The second condition says that there is no extension of a computation of a channel c between processes in Q after w. The two conditions, together, imply that the processes in Q cannot execute events after system computation w.

Example: Full-Buffer Deadlock Consider a system in which each channel is a buffer with a capacity of N, where $N > 0$. A process is either waiting or active. A waiting process is waiting to send a message on any one of a set of full outgoing channels (i.e., channels containing N messages); a waiting process continues to wait until at least one of the channels that it is waiting for becomes not full, and it then sends a message on that channel and becomes active. Waiting processes do not receive messages. A set of processes, Q, is said to be deadlocked if and only if:

1. each channel between processes in Q is full (or equivalently, the number of messages sent on the channel exceeds the number of messages received on the channel by N), and

2. each process in Q is waiting to send messages only along channels to other processes in Q.

The problem is to detect a deadlocked state.

A dual of this problem is obtained by replacing 'full' by 'empty', 'send' by 'receive', and 'outgoing' by 'incoming' in the previous problem.

Next, we give a theorem that shows how partial snapshots may be employed.

5.2 Termination Detection Theorem

Let v be a system computation such that Q terminates after v. If z is a system computation such that for all q in Q, $v.q$ is a prefix of $z.q$, then $v.q = z.q$, for all q in Q.

Proof We prove by induction on the length of prefixes u of z, that $\iota.q$ is a prefix of $v.q$, for all q in Q. In particular, we prove that $z.q$ is a prefix of $v.q$. Since $v.q$ is a prefix of $z.q$, it follows that $v.q = z.q$.

Base Case u is the empty sequence. The result holds, trivially.

Induction Step Consider a channel c used by processes q and r, where both q and r are in Q. Let $u.r = v.r$ (and $u.q$ is a prefix of $v.q$ from the induction hypothesis). From the monotonicity property, for all events

e on c, if the projection of (v, e) on c is not a computation of c, then the projection of (u, e) on c is not a computation of c. Since Q terminates after v, the projection of (v, e) on c is not a computation of c. Hence, if $u.r = v.r$, for all events e on c, the projection of (u, e) on c is not a computation of c. Since Q terminates after v, the only events on r after $v.r$ are events on channels to other processes in Q. Hence, if $u.r = v.r$, there is no event e on r such that (u, e) is a system computation.

From the arguments of the last paragraph, if (u, e) is a computation of z, then e is on a process r such that $u.r \neq v.r$. Since $u.r$ is a prefix of $v.r$, the length of $u.r$ is less than the length of $v.r$ In this case, $(u.r, e)$ is a prefix of $v.r$, since both $(u.r, e)$ and $v.r$ are prefixes of $z.r$, and the length of $(u.r, e)$ is at most the length of $v.r$.

5.3 Applications of the Theorem

The termination detection theorem tells us that old data $(v.q)$ is current (because $v.q = z.q$) if the old data shows that Q has terminated. This suggests the following class of algorithms for termination detection; this class includes algorithms in [1,4,9].

A Class of Algorithms for Termination Detection The algorithms employ a set of process computations $\{v.q | q \in Q\}$ and have the following specification.

Invariant $v.q$ is a prefix of $z.q$ where z is the system computation up to the current point.

Progress For all q, if the current value of $z.q$ is, say, $y.q$, then eventually $y.q$ is a prefix of $v.q$. (The progress property says that the process computations $v.q$ get updated: eventually, $v.q$ will include the current value, $y.q$, of $z.q$.)

The algorithm determines that Q has terminated if Q terminates after $\{v.q | q \in Q\}$, i.e., if Q terminates after a system computation, y, where $y.q = v.q$, all q in Q.

Correctness The proof of correctness of this class of algorithms is as follows. From the invariant and the theorem, if Q has terminated after $\{v.q|q \in Q\}$, then Q has terminated after z. From the progress property, if Q terminates after z, then eventually $v.q = z.q$, and hence eventually, the algorithm determines that Q has terminated.

Example Next, we give an example of algorithms with the invariant and progress properties given earlier. To detect termination of Q, a token is sent from process to process in Q, in such a manner that the token visits every process in Q repeatedly. The token carries with it a set of process computations $\{v.q|q \in Q\}$. Initially, $v.q$ is the empty sequence. When the token arrives at a process q, it updates this set, by replacing the value of $v.q$ in the set by its current computation, and q determines that Q has terminated if Q terminates after $\{v.q|q \in Q\}$.

Various optimizations are possible in applying this method to detect a specific form of termination. For example, to detect full-buffer deadlock, it is not necessary for the token to carry the entire computation $v.q$; it is sufficient for the token to contain the number of messages sent and received on each channel by q in $v.q$, and the set of processes for which q is waiting.

6 Summary

The paper presents necessary and sufficient conditions for a set of process computations to be a global snapshot. The condition is that for every channel, the computations in the snapshot of the processes that use the channel, are compatible with respect to the channel. The condition is helpful in the development of algorithms.

The paper also presents the concept of partial snapshots and demonstrates its utility.

References

[1] Chandy, K. M.[1987] 'A Theorem on Termination of Distributed Systems', TR-87-09, March 1987, Dept. of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712-1188.

[2] Chandy, K. M., and L. Lamport [1985]. 'Distributed Snapshots: Determining Global States of Distributed Systems,' ACM TOCS, 3:1, February 1985, pp. 63-75.

[3] Chandy, K. M. and J. Misra [1988] *Parallel Program Design: A Foundation*, Addison-Weslev, Reading, Massachusetts, 1988.

[4] Chandy, K. M. and J. Misra [1988] 'On Proofs of Distributed Algorithms with Application to the Problem of Termination Detection', submitted to Distributed Computing [1987].

[5] Dijkstra, E. W. [1985]. 'The Distributed Snapshot of K. M. Chandy and L. Lamport,' in Control Flow and Data Flow, ed. M. Broy, Berlin: Springer-Verlag, 1985, pp. 513-517.

[6] Fischer, M. J., N. D. Griffeth, and N. A. Lynch [1982]. 'Global States of a Distributed System,' IEEE Transactions on Software Engineering, SE-8:3, May 1982, pp. 198-202.

[7] Lamport,L.[1978] 'Time, Clocks and the Ordering of Events in a Distributed System,' C.ACM, 21:7, July 1978, pp 558-565.

[8] Hoare, C. A. R. [1984]. *Communicating Sequential Processes*, London: Prentice-Hall International, 1984.

[9] Raynal, M., J.-M. Helary, C. Jard, and N. Plouzeau [1987]. 'Detection of Stable Properties in Distributed Applications,' in Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, 1987, pp. 125-136.

To be published in *Proc 1989 ACM Symposium on Parallel Algorithms and Architectures*

A Framework for Adaptive Routing in Multicomputer Networks

John Y. Ngai and Charles L. Seitz
Department of Computer Science, California Institute of Technology
Submitted to the 1989 ACM Symposium on Parallel Algorithms and Architectures*

Introduction

Message-passing concurrent computers, also known as *multicomputers*, such as the Caltech Cosmic Cube [1] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes [2]. The communication networks of the second-generation machines, such as the Symult Series 2010 and the Intel iPSC2, employ an *oblivious* wormhole routing technique [3,4] that guarantees deadlock freedom. The message latency of this highly evolved oblivious technique has reached a limit of being capable of delivering, under random traffic, a *stable* maximum sustained throughput of $\approx$ 45 to 50% of the limit set by the network bisection bandwidth. Any further improvements on these networks will require an *adaptive* utilization of available network bandwidth to diffuse local congestions.

In an adaptive multipath routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. It is expected that such an adaptive control can increase the throughput capability towards the bisection bandwidth limit, while maintaining a reasonable network latency. While the potential gain in throughput is at most only a factor of 2 under random traffic, the adaptive approach offers additional advantages, such as the ability to diffuse local congestions in unbalanced traffic, and the potential to exploit inherent path redundancy in richly connected networks to perform *fault-tolerant* routing. The rest of this paper consists of an examination of the various feasibility issues and results concerning the adaptive approach studied by the authors.

*The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745, and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

A much more detailed exposition, including results on performance modeling and fault-tolerant routing, can be found in [5].

The Adaptive Cut-Through Model

It is clear that in order for the adaptive multipath scheme to compete favorably with the existing oblivious wormhole technique, it must employ a switching technique akin to *virtual cut-through* [6]. In cut-through switching and its blocking variant, which is used in oblivious wormhole routing, a packet is forwarded immediately upon receipt of enough header information to make a routing decision. The result is a dramatic reduction in the network latency over the conventional store-and-forward switching technique under light to moderate traffic. We now describe a simple cut-through switching model that provides the context for the discussion of issues involved in performing adaptive routing in multicomputer networks. The following definitions develop the notation that will be used throughout the rest of the paper.

Definition 1 A *Multicomputer Network*, M, is a connected undirected graph, $M = G(N, C)$. The vertices of the graph, N, represent the set of computing nodes. The edges of the graph, C, represent the set of *bidirectional* communication channels.

Definition 2 Let $n_i \in N$ be a node of M. The set, $C_i \subseteq C$, is the set of bidirectional channels connecting n_i to its neighbors in M.

Definition 3 The *width*, W, of a channel is the number of data wires across the channel. A *flit*, or *flow control unit*, is the W parallel bits of information transferred in a single cycle. The flit is the unit used to measure the length of a packet.

Definition 4 Given a pair of nodes, n_i and n_j, the set, Q_{ij}, of *routes* joining n_i to n_j is the fixed and predetermined set of directed *acyclic* paths from the source node, n_i, to the destination node, n_j.

Definition 5 For each destination node, n_j, the *profitable* channel set $R_{ij} \subseteq C_i$ is the subset of channels connected to n_i, where $c_k \in R_{ij} \Rightarrow c_k \in q_m \in Q_{ij}$. In other words, forwarding a packet along the routes in Q_{ij} is equivalent to sending it out through a profitable channel in R_{ij}.

Definition 6 For each node $n_i \in N$, the *Routing Relation* $R_i = \{(n_j, c_k) : n_j \in N - \{n_i\}, c_k \in R_{ij}\}$ defines for each possible destination node $n_j \in N$ its corresponding profitable channel set, R_{ij}.

Definition 7 The actual path a packet traverses while in transit in the communication network is referred to as the *trajectory* of the packet. Packet trajectories are identical to the packet routes in oblivious routing schemes but are *non-deterministic* in our adaptive formulation.

We assume the following:

- Long messages are broken into packets that are the logical data entities transferred across the network.
- Packets are of fixed length; *ie*, packet length $= L$, where L is a network-wide constant.
- Complete routing information is included in the header flit of each packet.
- Packets are forwarded in virtual cut-through style.
- A message packet arriving at its destination node is consumed. This is commonly known as the *consumption assumption*.
- A node can generate messages destined to any other node in the network.
- Nodes can produce packets at any rate subject to the constraint of available buffer space in the network, and packets are source queued.
- Each node in the network has complete knowledge of its own routing relation.

Figure 1 presents our view of the structure of a node in a multicomputer network. Conceptually, a node can be partitioned into a computation subsystem, a communication subsystem, and a message interface. For our purpose, the computation subsystem serves as the producer and consumer of the messages routed by the communication subsystem of the node. The message interface consists of dedicated hardware that handles the overhead in sending, receiving, and reassembling of message packets.

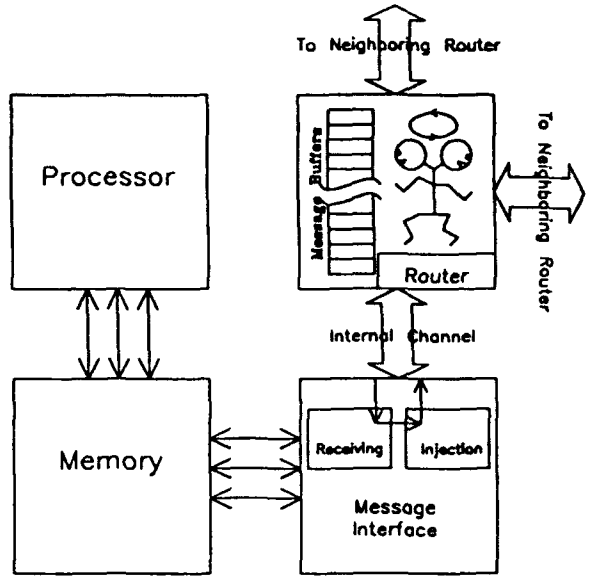


Figure 1: Structure of a node.

Internally, the communication subsystem consists of an adaptive control and a small number of message-packet buffers. Routing decisions are made by the adaptive control, based entirely on locally available information. The bidirectional channel assumption is adopted to allow the network to exploit locality in general message-communication patterns. Furthermore, it assures an identical number of input and output communication channels in each node, irrespective of the underlying network topology. The fixed-packet-length assumption is not essential and can be replaced by a *bounded*-packet-length assumption; *ie*, packet length $\leq L$, without invalidating any of our major results. It is adopted solely to simplify our subsequent exposition.

Communication Deadlock Freedom

In any adaptive routing scheme that allows arbitrary multipath routing, it is necessary to assure freedom from communication deadlock. Communication deadlock is caused generically by the existence of *cyclic* dependencies among communication resources along the message routes. Methods to prevent communication deadlock have been intensively researched and many schemes exist; of these, the methods of structured buffer pools [7] and virtual channels [8] are representative. In essence, all of these methods approach the problem by *re-mapping* any dependency that is potentially cyclic into a corresponding *acyclic* dependency structure. These methods employ restructuring techniques that require information of a global, albeit static, character. In contrast, a very simple technique that is *independent* of network size and topology, through vol-

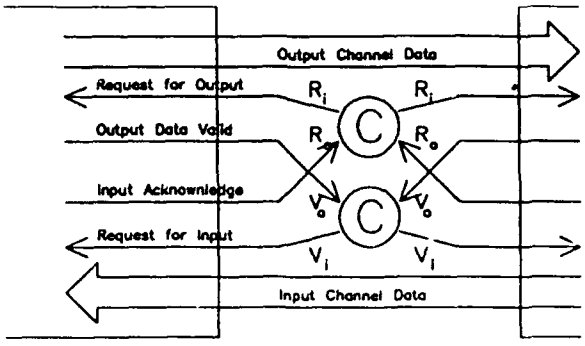


Figure 2: Two-phase protocol signaling.

untary *misrouting*, was suggested in [9] for networks that employ data exchange operations. Such a preemption technique utilizes only local information, and is dynamic in character. It prevents deadlock by *breaking* the potentially cyclic communication dependencies into disjoint paths of unit length. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly *matched*. A simple way is to have all bidirectional channels of the same node operate *coherently* under the protocol described next.

The Coherent Protocol. We now describe the channel data-exchange protocol in detail. It is used to match the transfer rates across all channels of the same node. The protocol employs four control signals per channel, two from each of the communicating partners, and is completely symmetric between the partners. The signaling events for a channel $c \in C$ are:

- R_o — output event to the communicating partner indicating that this node is Ready to accept another input flit from its partner. It also serves as an acknowledgment to its partner for the successful completion of the previous transfer cycle.

- R_i^c — input event from the communicating partner indicating that the partner is Ready to accept another output flit from this node. It is also an acknowledgment from the partner for the successful completion of the previous transfer cycle.

- V_o — output event to the communicating partner indicating that the data flit values currently held at the output channel of this node are Valid and its partner should latch in the held values.

- V_i^c — input event from the communicating partner indicating that the data flit values currently asserted at the input channel of this node are Valid and the node should latch in the held values.

We proceed to define our handshaking protocol across channels of a node $n_k \in N$, in a CSP-like notation [10]:

$$*[\ R_o;\ [\forall c \in C_k,\ R_i^c];\ \textit{apply out data};\quad V_o;\ [\forall c \in C_k,\ V_i^c];\ \textit{latch in data};\ ]$$

Observe that R_o and V_o denote, respectively, the *unique* outgoing Ready and data Valid signaling event to all neighbors of n_k. This enforces the matching of outgoing data rates. On the other hand, the matching of incoming data rates is enforced through the *synchronized wait* for the R_i^c and V_i^c signaling events from all neighbors. The handshaking events R_o, R_i^c interlock with the events V_o, V_i^c to guarantee the stability and strict alternation of each other. The *initial* state of a channel has both directions of the channel ready to accept a new data flit and proceeds thereafter in a *demand-driven* fashion. Figure 2 shows a possible conceptual realization of the protocol under the two-phase signaling convention [11] popular for off-chip communication. Since all the handshaking events defined are local between nearest neighbors, a network following the coherent protocol is *arbitrarily extensible*.

Observe that under cut-through switching, a packet can span many different channels. An outgoing channel occupied by a packet may not be able to assert V_o until after valid data has been asserted by the corresponding incoming channel occupied by the packet, hence, induces matching of data rates across the two occupied channels. The notion of coherency introduced here is a natural way to accommodate such potential dependencies among the various channels of a node under cut-through switching. Another notion that arises naturally is that of a *null flit*. To effect a transfer of data in one direction of a channel while the opposite direction is idle, the receiving partner is required to transmit a null flit in order to satisfy the convention dictated by the exchange protocol.

Deadlock Freedom. We now demonstrate that to assure communication deadlock freedom for networks operating under the coherent protocol, it is sufficient to employ voluntary misrouting to prevent potential buffer overflow. To proceed, observe that routing under the cut-through switching model imposes the following integrity constraints:

1. **Packets must always be forwarded to neighbors with their header flits transmitted first. In particular, voluntary misrouting of any internally buffered packet must start from the header flit of the selected packet.**

2. **Once the flit stream of a packet has been assigned a particular outgoing channel, the assignment must be maintained for the remaining cycles until the entire packet has been transmitted.**

These constraints exist because all of the necessary routing information of a packet is encapsulated in the packet header. Interrupting a packet flit stream mid-transfer would render the latter part of the packet undeliverable. To establish deadlock freedom, it is sufficient to show that each node can *independently* complete each transfer cycle and initiate a new one, in a bounded period, without violating the stated constraints. We now show that as long as we have an equal number of input and output channels per node, a condition satisfied readily by our bidirectional channel assumption, we can always satisfy the stated logical requirements, and, hence, assure freedom from communication deadlock.

Theorem 1 Let M denote a coherent multicomputer network where each node has an equal number of input and output channels. If M employs voluntary misrouting to prevent potential buffer overflow, then it is free from deadlock.

Proof. We need to show that buffer overflow can always be prevented by misrouting without violating the cut-through switching integrity constraints. We proceed with a counting argument: Let d denote the number of channels at a node. During a protocol cycle, there may be as many as $n^* \leq d$ new data flits arriving at the input channels. A fraction of these, $0 \leq n' \leq n^*$, are new header flits; the remaining $n^* - n'$ are non-header flits of arriving packets. Of these non-header flits, a fraction of them, $0 \leq n'' \leq n^* - n'$, belong to packets that have already been assigned output channels, and the remaining $n^* - n' - n''$ flits belong to waiting packets that are buffered inside the node. Therefore, the node has at least a total of $n' + (n^* - n' - n'')$ headers flits that are eligible for immediate routing. Hence, in the following cycle, a node can find at least $n' + (n^* - n' - n'') + n'' = n^*$ flits that can be transmitted or misrouted without violating the cut-through switching integrity constraints. This assures that no buffer overflow will occur. The node can always complete its protocol cycles in bounded time; hence, the network is free from deadlock. ■

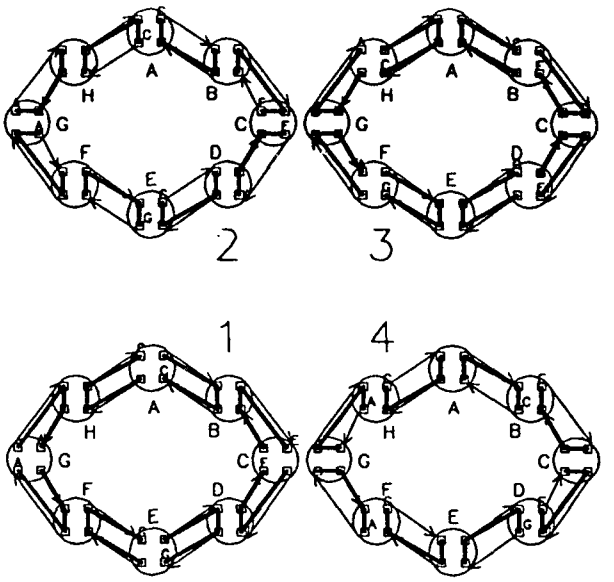


Figure 3: Livelock due to bad assignments.

Since the validity of the above proof does not depend on a node's storage capacity, deadlock freedom is established independent of the amount of available buffer space. The simple criterion of having an equal number of input and output channels is sufficient to assure deadlock freedom for a coherent network. In practice, additional buffers are needed in order to inject packets into the network, and to improve the network performance.

Network Progress Assurance

The adoption of voluntary misrouting renders communication deadlock a non-issue. However, misrouting also creates the burden to demonstrate progress in the form of message delivery assurance. In particular, a network can run into a *livelock*. Consider the sequence of routing scenarios depicted in figure 3 for a bidirectional ring consisting of eight nodes and eight packets. Each of the packets consists of four data flits that span multiple channels and internal buffers. Suppose the nodes employ the following simple, deterministic, packet-to-channel assignment rule: Whenever two incoming packets both request the same outgoing channel, the packet from the clockwise neighbor always wins. Given that, initially, nodes A, C, E, and G each receive two packets destined to nodes that are, respectively, distance two from them in the clockwise direction, after four routing cycles, the packets are all back to where they started! This example illustrates that packets can be forever denied delivery to their destinations even in the absence of communication deadlock.

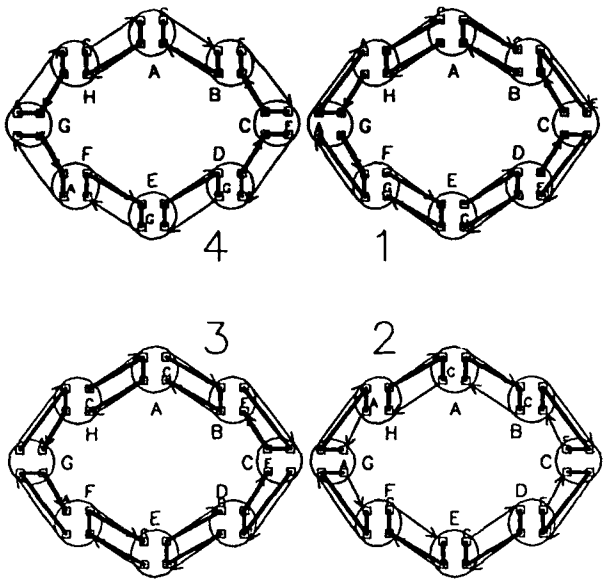


Figure 4: Livelock due to lack of assignments.

Channel-access competitions are, however, not the only type of conflict that can lead to livelock. Consider the situations depicted in figure 4 for the same bidirectional ring network. The traffic patterns are coincidental in such a way that none of the packets will ever have a chance to select its own output channel; rather, at every node, each packet must be forwarded along the only remaining channel, in compliance with the voluntary misrouting discipline, in order to avoid deadlock. It is clear that no matter what assignment strategy one chooses, it is impossible to break this kind of livelock without adding extra buffers per node. In other words, additional measures and resources have to be introduced in order to assure progress, *ie*, delivery of packets, in the network.

Buffering Discipline and Requirement. In order to assure packet delivery in spite of voluntary misrouting, extra buffers are required to store packets temporarily. In particular, sufficient buffers must be provided to allow the adaptive control to give *any* newly arriving packet a chance to escape preemption if so determined by the assignment algorithm. We now demonstrate the existence of such a solution using a bounded number of buffers. We assume the following buffering discipline:

1. Storage is divided into buffers of equal size; each is capable of holding an entire message packet.

2. Each buffer has exactly one input and one output port; this permits simultaneous reading and writing. A good example is a *FIFO* queue of length L.

3. Except as stated below, a buffer can be occupied by only one packet at a time. Oftentimes a packet may not fill its entire buffer, as in case of a partial cut-through. Such a packet occupies both the input and output ports to the buffer.

4. A buffer can be used temporarily to store two packets at a time, if and only if, one of them is leaving through the output port connected to an output channel, and the other is entering through the input port connected to an input channel.

Let b and d denote, respectively, the number of buffers and channels, *ie*, the *degree* at each node. First, we observe that, given the above buffering discipline, we must have $b \geq d$. To see this, assume that $L \gg d$, and consider the following sequence of events at a node with all buffers initially empty: At cycle $t = 0$, a packet P_0 arrives and is forwarded to its requested output channel c^* at cycle $t = 1$. Then, at cycles $t = L-d$ up to $t = L-2$, a total of $d-1$ packets, P_i, $i = 1, \ldots, d-1$, arriving one after another in these $d-1$ consecutive cycles, all requesting the same output channel c^*. Finally, at cycle $t = L+2$, another packet P_d arrives, requesting the same channel c^*. The worst case happens when the assignment algorithm always favors the latest arriving packet requiring it to stay and avoid preemption, and having each occupy a distinct buffer. Given the above arrival sequence, at cycle $t = L+1$, packet P_{d-1} will be forwarded through c^*, which now becomes idle. As a result, each packet from P_1 up to P_d would have to be temporarily stored as it comes in. Since each packet must be allocated to a distinct buffer, we must have $b \geq d$. We now show that having $b = d$ buffers is also sufficient.

Theorem 2 Let M be a coherent network where each node has b packet buffers inside the router operating under the stated assumptions. Then $b = d$ buffers per router is necessary and sufficient to always allow at least one packet, chosen arbitrarily by the assignment algorithm at each node, to escape preemption.

Proof. Necessity follows immediately from the preceding discussion. We proceed to establish sufficiency through a counting argument. Observe that a node is required to consider misrouting of packets in the next cycle only when there are new packets arriving at the current cycle. Figure 5 depicts an accounting of all possible cases of buffer allocation at the end of any such routing cycle. Let n_1 up to n_7 denote, respectively, the number of packets or buffers in each case; and n_0 denote the number of

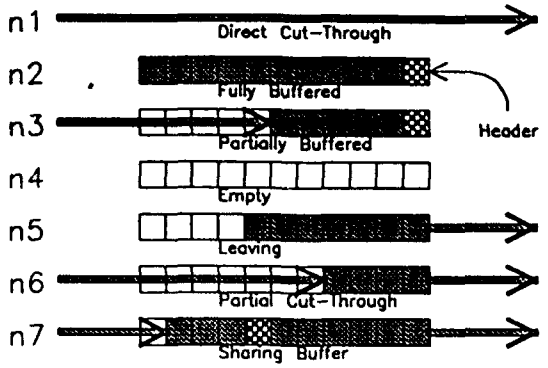


Figure 5: Accounting of buffer allocations.

newly arrived packets. Then, for inputs, we have $n_0+n_1+n_3+n_6+n_7 \leq d$; for outputs, we have $n_1+n_5+n_6+n_7 \leq d$. Let P^* denote the privileged packet chosen by the assignment algorithm to stay behind and avoid misrouting in the following cycle. P^* must be either a newly arrived packet or an already buffered packet. If P^* is a buffered packet, then a newly arriving packet either finds an idle output channel to directly cut through the node; or else we must have $n_1+n_5+n_6+n_7 = d \Rightarrow n_5 \geq n_0+n_3$, which, in turn, implies that there will always be an available buffer ready to accept it. On the other hand, if P^* is a newly arriving packet, then either $n_4+n_5 > 0$, and, hence, there is a buffer ready to accept it; or else we must have $n_2+n_3+n_6+n_7 = b = d$. This, together with the above inequality on inputs, $\Rightarrow n_2 \geq n_0+n_1 \Rightarrow n_2 > 0$. Furthermore, $n_0 > 0 \Rightarrow n_1+n_6+n_7 < d$. In other words, the packet will be able to find at least one buffer with a full idle packet as well as an idle output channel to preempt this idle packet and thus make room for itself. This establishes the sufficiency condition. ■

The trick in allowing the escape of misrouting for any arbitrarily chosen packet is to provide at least a critical, minimum number of buffers that is sufficient to assure either that empty buffers still exist, or that all buffers have been occupied, and, hence, there is some other packet that can be misrouted instead. The particular number required depends on the adopted buffering structure and discipline, and adding more buffers per node will allow the assignment algorithm to operate with more flexibility and perform better. In any case, by having a sufficient number of buffers, competition of profitable channel access is transformed into a competition for the right to stay behind and wait until the winner's profitable channel becomes available, at which time, it will be forwarded. Hence, winners that have been chosen by the assignment algorithm will have the chance to follow the actual paths determined by the routing relations. In a sense, assurance of packet delivery has now been reduced to that of picking *consistent* winners across the network.

Packet-Priority Assignments. An effective scheme for picking consistent winners that is independent of any particular network topology is to resolve the channel-access conflicts according to a *priority* assignment. In particular, the process of forwarding a packet towards its destination can be viewed as a sequence of actions performed to reduce the packet's distance from destination, provided that the set $\mathcal{R} = \{R_i\}$ of routing relations is defined in terms of an underlying metric of the network. In this case, as the result of a channel-access conflict, the winner will be routed along a profitable channel, hence decreasing its distance from the destination. The losers, depending on whether they are misrouted along the remaining unprofitable channels, may or may not increase their distance from destination. Ideally, one would prefer a strict monotonic decrease of distance to destination for each packet routed in the network. As this is impossible under our adaptive model, the alternative is to ensure monotonic decrease over a sequence of exchanges involving multiple packets. This can be achieved by giving higher priority to packets with shorter distances from destination over those with longer distances as follows:

$$P_1 > P_2 \Longleftrightarrow D_1 < D_2$$

where P is a packet's priority and D its distance from destination. We now show that this is sufficient to guarantee livelock freedom.

Theorem 3 A packet-to-channel assignment strategy that observes the defined distance priority, together with the set $\mathcal{R}$ of metric-based routing relations, guarantees livelock freedom in a network.

Proof. At the beginning of a routing cycle, let $D > 0$ be the minimum packet distance from destination. During this cycle, a packet with distance D competes with other packets for channels leading to its destination. If it wins the competition, it will be forwarded along a profitable channel within L cycles. It it loses, it must be to another packet also distance D away from its destination, according to the defined priority. In both cases, the minimum distance is reduced to $< D$ within L cycles. Therefore, D will eventually be reduced to zero, in which case a successful packet delivery occurs and the above argument can be applied again to assure repeated deliveries. This establishes livelock freedom. ■

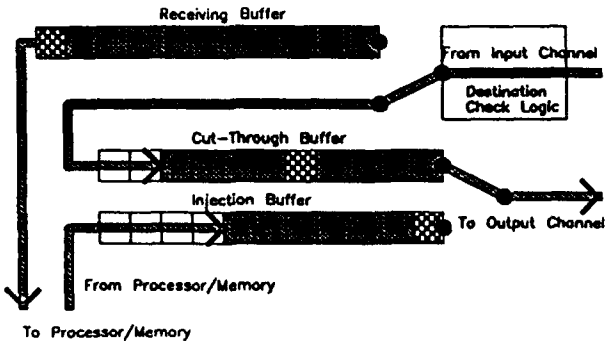


Figure 6: Inside the message interface.

Observe that although the distance priority alone suffices to guarantee global progress in a message network, no corresponding statement can be made concerning each individual packet. This is because it is possible for packets that are far away from their destinations to be repeatedly defeated by newly injected packets that are closer to their respective destinations. A more complex priority scheme that assures delivery of *every* packet can be obtained by augmenting the above simple scheme with *age* information, with higher priorities assigned to older packets:

$$(A_1, D_1) > (A_2, D_2) \iff (A_1 > A_2) \vee ((A_1 = A_2) \wedge (D_1 < D_2))$$

where A is a packet's *age*, that is, the number of routing cycles elapsed since the injection of the packet. Empirical simulation results indicate that the simple distance assignment scheme is sufficient for almost all situations, except under an extremely heavy applied load.

Network-Access Assurance

A different kind of progress assurance that requires demonstration under our adaptive formulation is the ability of a node to inject packets eventually. Because of the requirement to maintain strict balance of input and output data rates, a node located in the center of heavy traffic might be denied access to the network indefinitely. Figure 6 depicts a possible conceptual realization of a message interface. Its operation is similar to the register insertion ring interface described in [12]. It uses two FIFO buffers that can be connected to the output channel towards the network via a switch. Whenever the node has a packet to transmit, it loads the packet into the injection buffer as soon as the buffer becomes empty. When message traffic arrives from the network input channel, it passes through the destination check logic, which redirects any traffic destined to this node to the node memory. Any remaining passing traffic is loaded into the cut-through buffer, which is normally connected to the output channel. Whenever the cut-through buffer becomes empty, the control logic checks to see if there is an output packet waiting for injection. In such case, the switch is toggled so that the output channel is connected to the injection buffer and the injection proceeds. As the output packet is being forwarded, any passing traffic is loaded into the cut-through buffer. The switch connection is flipped back to the cut-through buffer after injection has been finished, and the process repeats. The main interesting property of the message interface for our current discussion is that it provides the mechanism to capture and accumulate interpacket gaps, which need not be contiguous, as empty spaces inside the cut-through buffers. When enough space has been collected, *ie*, the entire packet length, hence, an entire empty buffer, another new packet can be injected into the network. With such a mechanism, the question of assuring eventual packet injection is translated into that of assuring arrival of enough interpacket gaps whenever a node has a packet injection outstanding.

Round-Trip Packets. One simple way to assure network access is to have each packet delivered by the network be returned to its original sender upon arrival at its destination. Since each message interface starts with an empty injection buffer, consumption of its own round-trip packets will always restore its ability to inject the next source-queued packet. More sophisticated versions of such a scheme will use several cut-through buffers, and will demand that packets be returned only if the stock of empty cut-through buffers has been depleted below a predetermined threshold. In this way, the number of round-trip packets can be dramatically reduced when traffic is relatively moderate. Unfortunately, as traffic density increases, the population of round-trip packets also increases, thus further decreasing useful network bandwidth.

Packet-Injection Control. A different scheme that does not incur this overhead is to have the nodes maintain a bounded synchrony with neighbors on the total number of injections. Nodes that fall behind will, in effect, prohibit others from injecting until they catch up. We shall adopt the convention that a node having no packet to inject has a *null* packet queued up; *ie*, during each routing cycle, every node either has a null or real packet ready to inject or else is in the process of injecting a real packet. The null-packet convention is required to prevent quiescent nodes that do not have any packet to inject from blocking injections in the

active nodes. Our scheme is to introduce *local synchronization* among neighboring nodes such that the total number of packets injected by a node after each routing cycle will not differ by more than K, a positive constant, from those of its neighbors. We assume that each node explicitly maintains records of the total number of packet injections made by each of its neighbors, measured *relative to that of its own,* and that the information required to update these records in each node is exchanged on separate direct links between the message interfaces among neighbors. A node is allowed to inject its queued packet only if its own number of total injections is fewer than K packet injections ahead of its minimum neighbor. Nodes that are allowed to inject will examine their queued packets. Null packets are always injected by convention, whereas real packets are injected only if the injection mechanism described previously finds at least one empty buffer available to absorb the injection transient. We now show that, with eventual delivery of the packets already injected, this injection synchronization protocol establishes cooperation among the nodes to assure the eventual occurrence of empty cut-through buffers in the message interface for nodes that have real packets waiting for injection as permitted by the protocol.

Lemma 4 A node that has a packet waiting for injection that is permissible under the above injection protocol will eventually inject.

Proof. Observe that, by convention, if the pending packet is null, the node is able to inject immediately, so that the lemma is true vacuously. We now proceed to establish its validity for real packets. Suppose, to the contrary, that a particular node, $n \in N$, is blocked from injection indefinitely because the injection mechanism cannot accumulate sufficient empty buffer space to absorb the injection transient. Our injection protocol then dictates that its neighbors also will be blocked indefinitely from injecting. These, in turn, indefinitely block their neighbors, and so on. Given a finite network, all nodes are eventually blocked from any further injection, and eventually *no* new packet can enter the network. Given the eventual delivery guarantee for packets already injected, ultimately the network will be void of packets; at that point, the input channel to the interface of n will become idle, thus enabling it to resume the accumulation of empty spaces inside the cut-through buffer. Eventually, it will have collected enough spaces to enable the injection of its queued packet into the network. This contradicts the original indefinite blocking assumption of n, hence establishing the validity of the lemma. ■

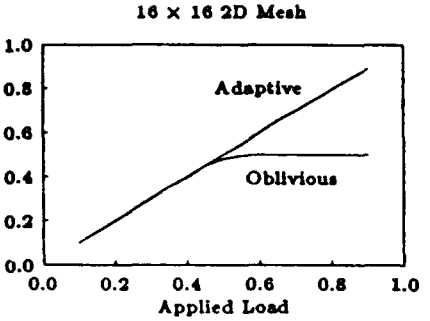


Figure 7: Throughput versus applied load.

We are now ready to show that by following the above injection protocol every individual node will eventually be permitted to inject, and, hence, according to the above lemma, will eventually inject. Specifically, let M be a network, and let T_i denote the total number of packet injections from node $n_i \in N$ since initialization. We now prove that T_i is strictly increasing over time.

Theorem 5 Given the injection protocol and a finite network that is livelock free, the total number of packet injections for each node strictly increases over time.

Proof. During a routing cycle, let $t = \min_{n_i \in N} T_i$ denote the minimum among numbers of packet injections since initialization, taken over all the nodes of the network, and let $S = \{n_i \in N | T_i = t\}$ denote the set of nodes that have recorded the minimum number of packet injections since initialization. Since $K > 0$, according to our protocol, every node $n \in S$ is permitted to inject. Lemma 4 then guarantees eventual injections from all of the nodes in S; hence, t, the minimum number of packet injections per node, is guaranteed to eventually increase over time. This, in turn, guarantees that T_i strictly increases over time, $\forall n_i \in N$. ■

Hence, we are assured of eventual packet injection for each individual node of the network. In other words, the above theorem establishes *fairness* in network access among all the nodes.

Performance Comparisons

An extensive set of simulations was conducted to obtain information concerning the potential gain in performance by switching from the oblivious wormhole to the adaptive cut-through technique. We now summarize very briefly the typical kind of behaviors observed in these simulations. A much more detailed discussion can be found in [5]. Among the

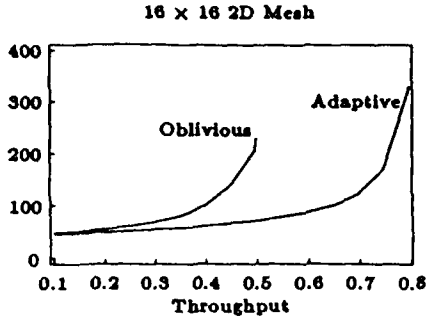


Figure 8: Message latency versus throughput.

various statistics collected, the two most important performance metrics in communication networks are *network throughput* and *message latency*. Figure 7 plots the sustained normalized network throughput versus the normalized applied load of the oblivious and adaptive schemes for a 16×16 2D-mesh network under random traffic. The normalization is performed with respect to the network bisection bandwidth limit. Starting at a very low applied load, the throughput curves of both schemes rise along a unit slope line. The oblivious wormhole curve levels off at $\approx 45 - 50\%$ of normalized throughput but remains *stable* even under increasingly heavy applied load. In contrast, the adaptive cut-through curve keeps rising along the unit slope line until it is out of the range of collected data. It should be pointed out, however, that the increase in throughput obtained is also partly due to the extra silicon area invested in buffer storage, which makes adaptive choices available.

Figure 8 plots the message latency versus normalized throughput for the same 2D-mesh network for a typical message length of 32 flits. The curves shown are typical of latency curves obtained in virtual cut-through switching. Both curves start with latency values close to the ideal at very low throughput, and remain relatively flat until they hit their respective transition points, after which both rise rapidly. The transition points are $\approx 40\%$ and 70%, respectively, for the oblivious and adaptive schemes. In essence, adaptive routing control increases the quantity of routing service, *ie*, network throughput, without sacrificing the quality of the provided service, *ie*, message latency, at the expense of requiring more silicon area.

Summary

Several issues related to adaptive cut-through routing have been addressed in the course of this research, and we did not encounter any insurmountable problem. Rather, the simplicity of these resolution mechanisms gives us hope that the adaptive scheme can be made to improve on the already highly evolved oblivious routing scheme. The discussion in this paper has focused on issues concerning the *feasibility* of the proposed adaptive routing framework. Within this framework, we have also studied and found promising approaches to fault-tolerant routing. Clearly, more work remains to be done. Perhaps the most challenging of all is to realise on *silicon*, the set of ideas outlined in this study.

References

[1] Charles L. Seitz, "The Cosmic Cube," *CACM*, 28(1), January 1985, pp. 22–33.

[2] William C. Athas and Charles L. Seitz., "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer*, August 1988, pp. 9–24.

[3] William J. Dally and Charles L. Seitz, "The Torus Routing Chip," *Distributed Computing*, 1986(1), pp. 187–196.

[4] Charles M. Flaig, *VLSI Mesh Routing Systems.* Caltech Computer Science Department Technical Report, 5241:TR:87.

[5] John Y. Ngai, *Adaptive Routing in Multicomputer Networks.* Ph.D. Thesis, Computer Science Department, Caltech. To be published.

[6] P. Kermani and L. Kleinrock, "Virtual Cut-Through : A New Computer Communication Switching Technique," *Computer Networks* 3(4) pp. 267–286, Sept. 1979.

[7] P. Merlin, and P. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks – I : Store-and Forward Deadlock," *IEEE Transactions on Communications*, Vol. COM-28, No. 3, pp. 345–354, March 1980.

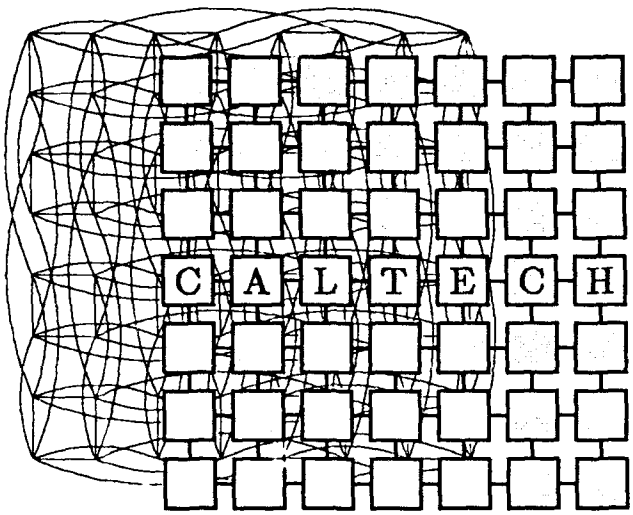
[8] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547–553, May 1987.

[9] A. Borodin, and J. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation," *Journal of Computer and System Sciences*, **30**, pp. 130–145 (1985).

[10] Alain J. Martin, "A Synthesis Method for Self-timed VLSI Circuits," *Proc. 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, IEEE Comp. Soc. Press, pp. 224–229 (1987).

[11] Charles L. Seitz, "System Timing," *Introduction to VLSI Systems*, C. Mead & L. Conway, Addison-Wesley, 1980, Chapter 7.

[12] M. T. Liu, "Distributed Loop Computer Networks," *Advances in Computers*, M. Yovits, Academic Press, pp. 163–221, 1978.



SUBMICRON SYSTEMS ARCHITECTURE PROJECT

Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-88-18

9 November 1988

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-18

9 November 1988

Reporting Period:	1 April 1988 – 31 October 1988 (7 months)
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	William C. Athas
	K. Mani Chandy
	Alain J. Martin
	Martin Rem
	Charles L. Seitz
	Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of the research activities and results for the seven-month period, 1 April 1988 to 31 October 1988, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Changes in Key Personnel

Dr. William C. Athas completed his appointment as a Postdoctoral Research Fellow in Computer Science in August 1988, and has joined the faculty at the University of Texas at Austin as an Assistant Professor of Computer Science. Dr. Stephen Taylor, a new PhD from the Weizmann Institute of Science and the author of a multicomputer implementation of flat concurrent prolog, joined the project in September 1988 with an appointment at Caltech as an Instructor in Computer Science.

2. Architecture Experiments

2.1 Mosaic Project

Bill Athas, Charles Flaig, Glenn Lewis, Jakov Seizovic, Don Speck, Wen-King Su, Tony Wittry, Chuck Seitz

The Mosaic C is an experimental multicomputer with single-chip nodes, currently in development. The stipulation that the nodes fit on a single chip so limits the storage for each node that relatively fine-grain concurrent programming techniques must be used. The Mosaic C will be programmed using the Cantor programming language, a fine-grain object-based (or Actor) language. We are working toward building a 16K-node Mosaic system using nodes fabricated in 1.2μm CMOS technology, with a near-term milestone of a 1K-node system using nodes fabricated in 1.6μm CMOS.

Much of our effort in this period has been concentrated on the Mosaic C project. The following is a brief summary of these activities (See also sections 3.1 & 4.5):

1. Cantor version 2.2 has been used internally within the research group for the past several months, and has been documented for external distribution. A technical report describing a collection of exemplary Cantor 2.2 programs that range up to 15 pages of program text in length was published. The report also reports the rationale for many of the design decisions in the evolution of Cantor from version 2.0 to 2.2.

2. Our initial implementation of a Cantor code generator for the Mosaic C indicated that only a simple procedure call mechanism was required; otherwise, the Mosaic C instruction set has been an efficient target for code generation. Work has commenced on a final Cantor code generator and runtime system for the Mosaic.

3. In accordance with the studies of code generation, the microcode for the Mosaic C processor was revised to implement an instruction set having a simpler procedure-call mechanism, together with several other minor refinements. The simplification of the instruction set reduced the number of implicants in the microcode that controls the processor from 66 to 102. The impact of this simplification on the processor area is merely favorable; its greatest benefit is in improving the processor speed (the RISC effect).

4. The entire processor was simulated at the clock-cycle and microcode level to debug and verify the microcode. The verified microcode was then used to generate a PLA structure, which was tied to the Mosaic C datapath for switch-level simulation and verification of the entire processor. A hybrid static/precharge PLA was designed to maximize the performance, and will be used in the final version of the processor.

5. An interface between the router and memory was designed, laid out, and verified by switch-level simulation. This final section of the Mosaic C single-

chip multicomputer node also includes the arbitration for memory refresh and memory access.

Fabrication of the first prototype processors and full Mosaic elements is now anticipated for early CY1989.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Alain Martin, Bill Athas, Charles Flaig, Jakov Seizovic, Craig Steele, Wen-King Su

Deliveries of the first production models of the Ametek Series 2010, a second-generation medium-grain multicomputer developed as a joint project between our research project and Ametek Computer Research Division, took place in this period. The reports we have received have been favorable. One customer who is also a DARPA contractor had developed 10,000+ lines of source code using the Cosmic Environment prior to taking delivery of the Ametek 2010, and apparently ported this code in a few days with no difficulties.

Additional benchmarks on the Ametek Series 2010 continue to show that it runs 8–10 times faster per node than such first-generation machines as the Intel iPSC/1.

Copies of the Cosmic Environment system have been distributed to approximately an additional 35 sites in this period, bringing the total copies distributed directly from the project to over 150. In addition, source copies of the Reactive Kernel node operating system were provided to two government contractors who are purchasing Ametek 2010 systems. An article titled "Multicomputers: Message-Passing Concurrent Computers" was published in the August 1988 issue of *IEEE COMPUTER*. This article on the current status of the multicomputers that have developed out of the work of our research group stimulated requests for many additional copies of "The C Programmer's Abbreviated Guide to Multicomputer Programming" [Caltech-CS-TR-88-1].

We expect to take delivery of the first 16-node increment of a 256-node Ametek 2010 in November 1988, and also a 16-node Intel iPSC/2, which will later be expanded to 64 nodes. Substantial blocks of time on the Ametek 2010 will be available to guest DARPA researchers.

Our Caltech project continues to work with both Ametek and Intel on the architectural design, message-routing methods and chips, and system software (evolutions of the Reactive Kernel (RK) node operating system and the Cosmic Environment (CE) host runtime system) for multicomputers. (See sections 3.2, 3.6 and 4.6 for details on these efforts.) We expect to see additional major advances in the performance and programmability of these systems over the next two years. In

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Ametek Computer Research Division (Monrovia, California).

addition, we continue to develop applications in VLSI design and analysis tools, and in other areas in which the programming of these multicomputer systems presents particular difficulties or opportunities. (See sections 3.3–3.5 and 4.9.)

2.3 Cosmic Cube Project

Bill Athas, Wen-King Su, Jakov Seizovic, Chuck Seitz

This section summarizes the current usage and the hardware and software status of our first-generation multicomputers, the Cosmic Cubes and Intel iPSC/1 d7.

These systems continue to operate reliably. Overall usage has been moderately heavy. The most time-consuming application in this period from within our own group has been a continuation of an extensive series of simulations by John Ngai concerned with the maximal utilization of networks with faulty routers or channels (see section ?). Supersonic flow computations being performed by students and faculty in Aeronautics at Caltech continue as the largest share of outside use.

The 64-node Cosmic Cube exhibited a hard failure in this seven-month period, a complete failure of its primary 5V, 130A power supply. The power supply was replaced, and the system rebooted without any problems. Counting the power supply failure as a single failure, the two original Cosmic Cubes have now logged 3.6 million node-hours with only four hard failures, three of them being chip failures in nodes. Curiously, we have not encountered a single connector failure. The calculated node MTBF of 100,000 hours reported before these machines were constructed was extremely conservative. A node MTBF in excess of 1,000,000 hours is probable, and can be stated at a 54% confidence level.

Our Intel iPSC/1 d7 (128 nodes) was contributed to the Submicron Systems Architecture Project as a part of the license agreement between the Caltech and Intel, and is accessible via the ARPAnet to other DARPA researchers who may wish to experiment with it. To request an account, please contact `chuck@vlsi.caltech.edu`. The Ametek Series 2010 system to be installed later this month will be available for outside use on a similar basis.

3. Concurrent Computation

3.1 Cantor

Nanette J. Boden, William C. Athas, Chuck Seitz

Programming for Fine-Grain Multicomputers

Over the last year we have been conducting a series of fine-grain programming experiments using Cantor. The purpose of this series of experiments was both to evaluate Cantor as a programming language and to investigate the nature of fine-grain programming. Application programs that have been written in these experiments include: fast-Fourier transform, shortest-path algorithms, a 2D convex hull solver, R-C chain-circuit simulation, digital logic simulation, a checkmate analyzer, an enumerator of paraffin isomers, and many others.

As a result of these programming experiments, modifications to Cantor have been made to facilitate fine-grain programming. Iteration internal to objects, custom objects, functional abstraction, and one-dimensional vectors are programming constructs that are now available in the newest version of Cantor, Cantor 2.2. A feature has also been added to the language to permit rudimentary discretion over message receipts. Analysis of the programming experiments clearly indicates that programming situations exist where some message discretion is very useful. In addition to these modifications, unnecessary features of the original language specification have been removed, including dynamic typing of variables. The changes that have been made to Cantor thus enhance programming abstraction while removing unnecessary constructs.

Using the latest version of Cantor as an experimental tool, we have written enough programs in the fine-grain style to draw some conclusions. Although formulations for Cantor programs are myriad, we have detected three general paradigms for the development of fine-grain programs:

1. Functional program specifications can be mapped directly into message-driven programs.
2. Solution specifications can be mapped into message-driven programs.
3. The object program can operate as a "logical apparatus" to solve the application problem.

In addition to observing these paradigms, we have been encouraged by the high degree of concurrency that is achieved in Cantor programs and by the convenience and generality of fine-grain programming. Based on our experiments with Cantor thus far, we believe that large, highly concurrent programs can be efficiently expressed in the fine-grain programming style.

Programming for the Mosaic

Recent research in the area of Mosaic programming has focused on the definition and analysis of an abstract machine for the execution of Cantor code. The Cantor Abstract Machine (CAM) definition is based on the fine-grain multicomputer architecture, yet encapsulates operations like object creation, message sends and receives, *etc,* in single instructions. The purpose of this approach is to isolate the implementation of these complicated operations as much as possible from the development of an efficient runtime system.

A new Cantor code generator and simulator have been written for the CAM. Analysis of the abstract machine has already suggested improvements in the Cantor intermediate format. In addition, simulation of program execution on the CAM is expected to be very useful in evaluating potential Mosaic runtime system alternatives.

3.2 The Cosmic Environment and Reactive Kernel

Jakov Seizovic, Wen-King Su, Chuck Seitz

The Cosmic Environment and Reactive Kernel continue to run reliably on the original Cosmic Cubes and on the Ametek Series 2010, and no major changes have been made. The internals of RK are now documented in technical report Caltech-CS-TR-88-10.

In the original version of the RK, we were able to guarantee the *weak fairness* of scheduling on a multicomputer node only if all processes on that node satisfied the reactive property that they would eventually either terminate, or execute an `xrecv()`. The producers of an infinite number of messages are an important class of processes that do not satisfy the reactive property. A simple modification of the implementation of the `xmalloc()` system call has enabled us to support the infinite computations as well. The `xmalloc()` system call is implemented in terms of the RPC mechanism. The requested buffer is not delivered immediately; instead it is sent to the requesting process and delivered through the regular scheduling mechanism.

3.3 CONCISE — A Concurrent Circuit Simulator*

Sven Mattisson, Lena Peterson, Chuck Seitz

Within this project, a concurrent circuit simulation program called CONCISE has been developed. This program is a circuit simulator for transient analysis of CMOS-circuits. It is written in C and uses the Cosmic Environment/Reactive Kernel message-passing primitives.

* This segment of our research is a joint project between the Caltech Submicron Systems Architecture Project and the Department of Applied Electronics at the University of Lund, Sweden.

Recently, CONCISE was ported to the Ametek Series 2010. Thus, the program now runs on several multicomputers with loosely coupled nodes, including the Ametek 2010 and the Intel iPSC, and on a shared memory multicomputer, the Sequent Symmetry. The port to the Ametek 2010 showed that CONCISE is more than eight times faster on the Ametek 2010 than on the Intel iPSC/1, which is a typical first-generation multicomputer.

The Reactive Kernel primitives support a programming model where each process has its own memory space. This model makes dynamic partitioning and load balancing expensive in CPU time. Thus, we have developed a static partitioning scheme that tries to enhance the convergence rate of the waveform relaxation method without sacrificing the grain-size of the computational tasks. It is important to notice that the requirements on the partitioning algorithms in this case differ from the "traditional" parallelization, where only a few processing nodes are used.

So far, six different combinations of iteration schemes and partitioning have been tested. The iteration schemes tested are ordinary Jacobi iterations, ordinary Gauss-Seidel, and n-colored Gauss-Seidel. The n-colored Gauss-Seidel uses the incidence-degree algorithm to find a coloring with the least number of colors for the circuit graph. Then, the different colors can be solved concurrently, since each node has a color different from those of its neighbors. These three algorithms have all been run with two different partitioning schemes: one in which each circuit node forms a cluster on its own, and one where source-drain connected circuit nodes are clustered together.

The results show that regular Gauss-Seidel iterations are not suitable except for very few processing nodes, and this scheme is the most popular for sequential waveform-relaxation implementations. Instead, the n-coloring version of Gauss-Seidel iterations are useful for the case when the number of processing nodes is large, but significantly less than the number of processes. The number of colors needed usually lies between three and five.

When the number of computing nodes is close to the number of circuit nodes, Jacobi iterations do surprisingly well. This is due to the fact that the load imbalance gets increasingly severe for the other schemes. For some circuits, the clusters get very big, and splitting schemes fail in producing reasonable size clusters that still achieve comparable convergence speed. For such circuits a hierarchical approach where more than one node can be assigned to solving a cluster would be desirable. Such an approach will be possible with the faster message passing of the second-generation multicomputers, and experiments in this area are presently being carried out.

In another effort, Concise has been used by Anthony Skjellum in the Chemical Engineering Department at Caltech for the simulation of distillation columns. This work has shown that it is possible to use Concise to simulate dynamic systems that

are not at all like circuits. As part of this effort, Concise has been modified to make it easier to install models of other kinds of "devices."

3.4 Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm

Wen-King Su, Chuck Seitz

A new and more versatile logic simulator has been written in the past six months to better evaluate a more diverse set of conservative variants of the Chandy-Misra-Bryant (CMB) distributed discrete-event simulation algorithm. Most of the conclusions from this study are included in the paper "Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm," accepted for publication in the 1989 SCS Eastern Multi-conference. The primary conclusions are that the variants examined are similar, in that all of them take an initial penalty running on a single node in comparison with sequential event-driven simulators that exploit an ordered event list. The penalty is due to the generation and the processing of null messages. However, as the number of processing nodes increases, the simulation time decreases linearly until all usable concurrency has been exhausted. Depending on the circuit being simulated, the crossover point (the point at which the time taken by the concurrent simulators drops below the time taken for the sequential simulator) has been observed to be anywhere between four and 200 nodes.

After the paper was submitted, a new simulator variant was written to try to reduce the initial overhead by combining sequential simulation methods with the concurrent simulator variants. The resulting simulator has the performace of a sequential simulator for the single processor case, and it converges with that of the concurrent simulator when the number of nodes is sufficiently large. However, the nature of the logic circuit being simulated strongly influences the rate of convergence. We have observed all three cases:

1. The simulation time humps upward toward that of the concurrent simulators as soon as the number of processing nodes is increased beyond one.
2. The simulation time remains the same until the concurrent-sequential crossover point.
3. The simulation time starts to decrease as soon as the number of nodes are increased, but the drop is less than linear.

A conclusion of this study is that very-high-performance logic simulation on concurrent computers is completely plausible for systems with very large numbers of nodes, where the CMB null-message scheme is fully exploited. Conversely, it is efficient for small-N systems only when the elements being simulated are more complex and have longer running times than logic elements.

3.5 Automatic Mapping of Processes and Channels

Drazen Borkovic, Alain Martin

To facilitate programming of message-passing machines, we have developed a preprocessor, **map**^{3}, that allows for a certain level of abstraction in the mapping of processes and channels on the nodes and physical channels of a message-passing multicomputer.

The description of a set of processes and the channels between them has been compiled into a set of C functions that perform the mapping of the processes onto physical nodes of the target machine. The preprocessor supports a hierarchical organization of processes and local names for the channels. There is also a set of library routines that can emulate channels with arbitrary slack.

The preprocessor and the library routines have been successfully implemented and tested under the Cosmic Environment/Reactive Kernel system.

3.6 A Multicomputer "Page Kernel"

Craig S. Steele, Chuck Seitz

As described in a previous report, an experimental "page kernel" is being developed that uses memory-access-protection mechanisms as the interface to multicomputer message subsystems. A prototype of the "page kernel" is now running on a sequential machine. The current code is simulating the memory-management hardware of the Ametek Series 2010 computing node, and will be ported to the Series 2010 shortly.

The page kernel supports dynamic load-balancing and process relocation. The kernel's ability to transparently update copies of data distributed across a multi-node system is particularly well-suited for chaotic iterative programs, such as process-placement optimization.

4. VLSI Design

4.1 Testing Self-Timed Circuits

Pieter Hazewindus, Alain Martin

We are investigating methods to test self-timed circuits. Traditionally, it is thought that these circuits are hard to test because of the possibility of races and hazards, and because these circuits are sequential. In our design method, however, races and hazards are absent.

The fault model we use is the stuck-at model, where each wire may be stuck forever at a high (logic-1) or low (logic-0) voltage. We have proven that it is sufficient to perform a single four-phase handshake on each channel to detect all detectable stuck-at faults. Some faults are undetectable.

For the automatic compilation, the main sequencing element is the so-called D-element. For the D-element, there are twenty-two possible stuck-at faults, two of which are undetectable. We have designed an alternate D-element that does not have any undetectable stuck-at faults. Most other circuit constructs in this compiler are completely testable.

Although it is not yet certain whether all constructs can be made entirely testable, our present estimate is that self-timed circuits designed according to our method should be easier to test than traditional clocked circuits.

4.2 A Self-Timed $3x + 1$ Engine

Tony Lee, Alain Martin

We have designed and fabricated a self-timed special-purpose processor for implementing the $3x+1$ algorithm. The processor consists of a state-machine and an 80-bit-wide datapath. It contains approximately 40,000 transistors and operates at over 8 MIPS in 2μm MOSIS SCMOS technology. As usual, the chip was functional on first silicon.

4.3 Performance Analysis of Self-Timed Circuits

Steve Burns, Alain Martin

We have developed methods for determining the repetition time of a set of communicating sequential processes described as handshaking expansions. This performance measure is provided in the form of constraint equations involving symbolic values of the communication and sequencing delays. The analysis is valid regardless of the actual delay values, and thus provides a means of comparing designs described at the handshaking expansion level without first generating detailed circuit implementations. Circuits for handshaking expansions that result in slow repetition times need never be designed.

This method has proven particularly useful in the analysis of programs involving data. It has been used throughout the design of the self-timed microprocessor, increasing the performance of programs involving data up to a factor of two.

4.4 The Design of a Self-Timed Microprocessor

Alain Martin, Steve Burns, Tony Lee, Drazen Borkovic, Pieter Hazewindus

In order to refute the claims that our design method would be too slow and too wasteful in area for anything but small circuits, we have embarked on the design of complete general-purpose microprocessor. The instruction set is "classic": 16-bit instructions with offset, load/store type of instructions, and separate memories for instructions and data. The only restriction is the absence of an interrupt mechanism.

As expected, since the method is based on concurrent programming techniques, the design is highly concurrent. The fetch, decode, and execute phases overlap, as do the execution of ALU and memory instructions. The different processes share 16 general-purpose registers, and four buses are used to communicate with the registers, in addition to point-to-point channels.

We are now in the layout phase of the design. Preliminary estimates of the performance are encouraging. In 2μm SCMOS, we expect to reach 20MIPS.

4.5 Mosaic Elements

Chuck Seitz, Bill Athas, Charles Flaig, Glenn Lewis, Don Speck, Jakov Seizovic, Wen-King Su

With the completion of the packet interface section and the near-completion of the processor, and with the other sections having already been fabricated and tested, the Mosaic C single-chip multicomputer node is rapidly approaching completion. Assembly of the sections will start within the next month, and fabrication of complete elements early in 1989.

The packet interface for the Mosaic chip has been layed out and verified with the switch-level simulation. It is entirely synchronous, and was designed conservatively, so no problems with it are anticipated.

The packet interface consists of two independent finite-state machines, one for sending packets, and the other for receiving packets. Both machines act as simple DMA channels, stealing unused memory cycles, and the packet interface is designed to be able to sustain a throughput equal to the maximum possible message rate that can be achieved by the message router.

The packet interface provides for a fairly complete testing of itself and the router, initiated by a CPU request to send a message to itself. In this mode of operation, the message will be taken from the memory, sent through all three router dimensions, and received back into the memory.

4.6 Fast Self-Timed Mesh Routing Chips

Charles Flaig, Chuck Seitz

A new design of a mesh routing chip (MRC), the FMRC2.0 design, was sent to fabrication in May 1988, together with a separate test chip containing only the FIFO used in the FMRC2.0. These chips employ a circuit design style that is potentially faster but less conservative than is usual for self-timed designs. The chips returned from fabrication do indeed operate nearly three times faster than previous designs. The FIFC test chip, fabricated in a 2μm MOSIS SCMOS process (this chip was also a test of the new 40-pin 2μm pads and design frame that we developed for MOSIS) operated correctly at 70 MBytes/s!

The critical path in a routing chip includes somewhat longer delay paths due to the switching of the packets; hence, although the FMRC2.0 was fabricated in a 1.6μm process, and its FIFOs might be expected to operate at around 85 MBytes/s, it operates as anticipated at 70 MBytes/s. However, it routes packets incorrectly, showing symptoms of directing packets according to the tail of the previous packet rather than the head of the current packet. This fault was finally traced to a timing error of approximately 0.7ns in the latching of a routing decision. The timing error was fixed, and the timing margins in the entire chip were reexamined. A *post facto* Spice simulation of what the analysis showed were the critical points in the old and new designs verified that the original design had a timing error of 0.7ns, while the revised design has a timing margin of about 1.0ns (about 50% of the difference between two short delay paths; hence, not as close as it may sound).

If successful, we expect this new FMRC chip to replace the MRC currently used in the Ametek Series 2010 multicomputer. With help from George Lewicki, this design is also being transferred to an Intel fabrication process for possible use in a future Intel multicomputer.

Tests of the self-timed FIFO in a 2μm MOSIS SCMOS technology will be of interest to other chip designers in the DARPA VLSI community — particularly those designing self-timed chips.

The 2μm FIFO tests yielded a request $\rightarrow$ acknowledge time of 6.5-7.0ns, and a throughput of over 70 MBytes/s on these byte-wide channels. Lest someone interpret this test result as implying that we are driving 70MHz signals through these pads, please understand that in 2-cycle R/A signaling (*cf,* Mead & Conway, figure 7.16), only one *transition* is required for each data transfer, so the maximum fundamental frequency on any R/A or data pin is 35MHz to transfer data at a 70MHz rate.

The total fall-through time for all 101 FIFO stages was measured as 350ns, or 3.5ns fallthrough per stage. The fallthrough time calculated by the τ-model is about 70τ, so this is consistent with a value of τ for the 2μm MOSIS SCMOS n-well process of about 50ps (which is a bit smaller than expected). The *internal*

cycle time when the operation is not impeded by signals passing through pads and package pins is about 180τ, or about 9ns, corresponding to an internal throughput rate of 114MHz.

These speeds in the 2μ MOSIS n-well SCMOS technology are, as expected, about twice as fast as a nearly identical test device fabricated in a 3μm MOSIS p-well SCMOS process. The fallthrough times are more difficult to measure in the 1.6μm FMRC2.0 chip, because of switching and address-decrementing logic in the FIFO pipeline. We can infer than the FIFO fall-through times are about 2.8ns per stage, corresponding to a τ of 40ps, and an internal throughput rate of about 140 MHz.

It is quite evident from these tests that we are able to achieve much higher internal speeds with self-timed and/or asynchronous designs than we know how to achieve with clocked designs.

4.7 Adaptive Routing in Multicomputer Networks

John Y. Ngai, Chuck Seitz

Our studies of adaptive routing in multicomputer networks are approaching a conclusion, and have been generally successful. We now believe that the Adaptive Cut-Through (ACT) routing scheme is capable of outperforming the existing highly evolved oblivious routing devices by a factor of about two in throughput, and have numerous other advantages in hot-spot throughput and fault-tolerance. A summary of the results of our investigations is attached at the end of this report.

What remains to be done to realize the advantages of the ACT routing scheme is to design a VLSI routing chip and/or a new routing section for the Mosaic C.

4.8 Pads and Pad Frame Generation

Charles Flaig, Chuck Seitz

Derived in large part from the pads and pad frames we have designed for mesh routing chips (MRCs), a variety of new pad circuits have been designed for the $\lambda = 0.6\mu$m, 0.8μm, and 1.0μm MOSIS SCMOS processes. One of these design variations was used to produce a new 2μm 40-pin "tiny-chip" frame for MOSIS, including input, Schmitt input, output, and tristate output pads. The unusual features of these pad designs include the use of longitudinal (bipolar) clamp transistors for static and overvoltage protection, and a variety of pad pitches.

We can now report some test results for the 2μm pads. This 40-pin pad frame was fabricated with a 101-stage self-timed FIFO from the FRMC2.0 design (see section 4.6), together with some output pads being driven directly from input pads.

Overvoltage clamping on the inputs clamps to 6V at 200mA, and 7V at 800mA, which is excellent. Undervoltage protection is about the same as above, BUT, at

about -500mA the chip appears to suffer latchup (if power is supplied). This is not a problem for normal static, where no Vdd is applied, but if an input does goes more than about 1V negative while power is applied, latchup may be induced.

For the Schmitt input pad, trigger voltages are 0.8V and 3.9V, for a 2.9V hysteresis. Inpad → Outpad delay is 1.5–2.0ns for no load, 2.0–2.5ns for a fanout of 1, and 2.5-3.5ns for a fan-out of 2. Rise/fall time is 3.5ns for no load, 4.5ns for a fanout of 1, and 6.5ns for a fan-out of 2. The output pads can sink about 30mA at 1.0V, or source about 30mA at 4.0V, under 5.0V operation. These characteristics are more than adequate for student projects.

4.9 The Notorious CIF-flogger Program

Glenn Lewis, Chuck Seitz

The CIF-flogger is a multicomputer program for flattening CIF files, rasterizing the geometry, and for performing parallel operations on the geometry in strips. It runs under the CE/RK system, and hence, on most available multicomputers, including the Ametek Series 2010.

The CIF-flogger currently supports simple bloat, shrink, and logical operations on the flattened geometry, and hence can perform most geometrical design-rule checks. It establishes connected component labeling and will eventually provide complete design-rule checking, well checks, and circuit extraction. Based on timings on the iPSC/1, CIF-flogger is expected to perform design rule checks for 100K-transistor chips in much less than 1s per rule on second-generation multicomputers.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports
23 August 1988
Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

der reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not cepted. All foreign orders must be paid by international money order or by check drawn on a U.S. bank in U.S. currency, ɔle to CALTECH.

ɔS-TR-88-17	$3.00	*Constrained Diffrential Optimization for Neural Networks*, Platt, John C and Alan H Barr
ɔS-TR-88-16	$3.00	*Programming Parallel Computers*, Chandy, K. Mani
ɔS-TR-88-15	$13.00	*Applications of Surface Networks to Sampling Problems in Computer Graphics*, PhD Thesis Von Herzen, Brian
ɔS-TR-88-14	$2.00	*Syntax-directed Translation of Concurrent Programs into Self-timed Circuits* Burns, Steven M and Alain J Martin
ɔS-TR-88-13	$2.00	*A Message-Passing Model for Highly Concurrent Computation*, Martin, Alain J
ɔS-TR-88-12	$4.00	*A Comparison of Strict and Non-strict Semantics for Lists*, MS Thesis Burch, Jerry R
ɔS-TR-88-07	$3.00	*The Hexagonal Resistive Network and the Circular Approximation*, Feinstein, David I
ɔS-TR-88-06	$3.00	*Theorems on Computations of Distributed Systems*, Chandy, K Mani
ɔS-TR-88-05	$3.00	*Submicron Systems Architecture* ARPA Semiannual Technical Report
ɔS-TR-88-01	$3.00	*C Programmer's Abbreviated Guide to Multicomputer Programming*, Seitz, Charles, Jakov Seizovic and Wen-King Su
5258:TR:88	$3.00	*Submicron Systems Architecture* ARPA Semiannual Technical Report
5256:TR:87	$2.00	*Synthesis Method for Self-timed VLSI Circuits*, Martin, Alain current supply only: see *Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design*, 224-229, Oct'87
5253:TR:88	$2.00	*Synthesis of Self-Timed Circuits by Program Transformation*, Burns, Steven M and Alain J Martin
5251:TR:87	$2.00	*Conditional Knowledge as a Basis for Distributed Simulation*, Chandy, K. Mani and Jay Misra
5250:TR:87	$10.00	*Images, Numerical Analysis of Singularities and Shock Filters*, PhD Thesis Rudin, Leonid Iakov
5249:TR:87	$6.00	*Logic from Programming Language Semantics*, PhD Thesis Choo, Young-il
5247:TR:87	$6.00	*VLSI Concurrent Computation for Music Synthesis*, PhD Thesis Wawrzynek, John
5246:TR:87	$3.00	*Framework for Adaptive Routing* Ngai, John Y and Charles L. Seitz
5244:TR:87	$3.00	*Multicomputers* Athas, William C and Charles L Seitz
5243:TR:87	$5.00	*Resource-Bounded Category and Measure in Exponential Complexity Classes*, PhD Thesis Lutz, Jack H

Caltech Computer Science Technical Reports

CR:87 $8.00 *Fine Grain Concurrent Computations*, PhD Thesis
Athas, William C.

CR:87 $3.00 *VLSI Mesh Routing Systems*, MS Thesis
Flaig, Charles M

CR:87 $2.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

CR:87 $3.00 *Trace Theory and Systolic Computations*
Rem, Martin

CR:87 $7.00 *Incorporating Time in the New World of Computing System*, MS Thesis
Poh, Hean Lee

CR:86 $4.00 *Approach to Concurrent Semantics Using Complete Traces*, MS Thesis
Van Horn, Kevin S.

CR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

CR:86 $3.00 *High Performance Implementation of Prolog*
Newton, Michael O

CR:86 $3.00 *Some Results on Kolmogorov-Chaitin Complexity*, MS Thesis
Schweizer, David Lawrence

CR:86 $4.00 *Cantor User Report*
Athas, W.C. and C. L. Seitz

CR:86 $24.00 *Monte Carlo Methods for 2-D Compaction*, PhD Thesis
Mosteller, R.C.

CR:86 $4.00 *anaLOG - A Functional Simulator for VLSI Neural Systems*, MS Thesis
Lazzaro, John

TR:86 $3.00 *On Performance of k-ary n-cube Interconection Networks*,
Dally, Wm. J

TR:86 $18.00 *Parallel Execution Model for Logic Programming*, PhD Thesis
Li, Pey-yun Peggy

TR:86 $15.00 *Integrated Optical Motion Detection*, PhD Thesis
Tanner, John E.

TR:86 $3.00 *Sync Model: A Parallel Execution Method for Logic Programming*
Li, Pey-yun Peggy and Alain J. Martin
current supply only: see *Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86*

TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

TR:86 $2.00 *How to Get a Large Natural Language System into a Personal Computer*,
Thompson, Bozena H. and Frederick B. Thompson

TR:86 $2.00 *ASK is Transportable in Half a Dozen Ways*,
Thompson, Bozena H. and Frederick B. Thompson

TR:86 $2.00 *On Seitz' Arbiter*,
Martin, Alain J

TR:86 $2.00 *Compiling Communicating Processes into Delay-Insensitive VLSI Circuits*,
Martin, Alain
current supply only: see *Distributed Computing* v 1 no 4 (1986)

TR:86 $2.00 *Complete and Infinite Traces: A Descriptive Model of Computing Agents*,
van Horn, Kevin

TR:85 $2.00 *Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity*,
Schweizer, David and Yaser Abu-Mostafa

TR:85 $3.00 *An Inverse Limit Construction of a Domain of Infinite Lists*,
Choo, Young-Il

TR:85 $15.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5200:TR:85 $18.00 *ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation*, PhD thesis
Whelan, Dan

5198:TR:85 $8.00 *Neural Networks, Pattern Recognition and Fingerprint Hallucination*, PhD thesis
Mjolsness, Eric

5197:TR:85 $7.00 *Sequential Threshold Circuits*, MS thesis
Platt, John

5195:TR:85 $3.00 *New Generalization of Dekker's Algorithm for Mutual Exclusion*,
Martin, Alain J
current supply only: see *Information Processing Letters*, **23**, 295-297 (1986)

5194:TR:85 $5.00 *Sneptree - A Versatile Interconnection Network*,
Li, Peiyun Peggy and Alain J Martin

5193:TR:85 $2.00 *Delay-insensitive Fair Arbiter*
Martin, Alain J
current supply only: see *Distr Computing* **1**:226-234 (1986)

5190:TR:85 $3.00 *Concurrency Algebra and Petri Nets*,
Choo, Young-il

5189:TR:85 $10.00 *Hierarchical Composition of VLSI Circuits*, PhD Thesis
Whitney, Telle

5185:TR:85 $11.00 *Combining Computation with Geometry*, PhD Thesis
Lien, Sheue-Ling

5184:TR:85 $7.00 *Placement of Communicating Processes on Multiprocessor Networks*, MS Thesis
Steele, Craig

5179:TR:85 $3.00 *Sampling Deformed, Intersecting Surfaces with Quadtrees*, MS Thesis,
Von Herzen, Brian P.

5178:TR:85 $9.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5174:TR:85 $7.00 *Balanced Cube: A Concurrent Data Structure*,
Dally, William J and Charles L Seitz

5172:TR:85 $6.00 *Combined Logical and Functional Programming Language*,
Newton, Michael

5168:TR:84 $3.00 *Object Oriented Architecture*,
Dally, Bill and Jim Kajiya

5165:TR:84 $4.00 *Customizing One's Own Interface Using English as Primary Language*,
Thompson, B H and Frederick B Thompson

5164:TR:84 $13.00 *ASK French - A French Natural Language Syntax*, MS Thesis
Sanouillet, Remy

5160:TR:84 $7.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5158:TR:84 $6.00 *VLSI Architecture for Sound Synthesis*,
Wawrzynek, John and Carver Mead

5157:TR:84 $15.00 *Bit-Serial Reed-Solomon Decoders in VLSI*, PhD Thesis
Whiting, Douglas

5147:TR:84 $4.00 *Networks of Machines for Distributed Recursive Computations*,
Martin, Alain and Jan van de Snepscheut

5143:TR:84 $5.00 *General Interconnect Problem*, MS Thesis
Ngai, John

5140:TR:84 $5.00 *Hierarchy of Graph Isomorphism Testing*, MS Thesis
Chen, Wen-Chi

5139:TR:84 $4.00 *HEX: A Hierarchical Circuit Extractor*, MS Thesis
Oyang, Yen-Jen

5137:TR:84 $7.00 *Dialogue Designing Dialogue System*, PhD Thesis
Ho, Tai-Ping

R:84 $5.00 *Heterogeneous Data Base Access*, PhD Thesis
Papachristidis, Alex

R:84 $7.00 *Toward Concurrent Arithmetic*, MS Thesis
Chiang, Chao-Lin

R:84 $2.00 *Using Logic Programming for Compiling APL*, MS Thesis
Derby, Howard

R:84 $13.00 *Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems*, PhD Thesis
Lin, Tzu-mu

R:84 $10.00 *Switch Level Fault Simulation of MOS Digital Circuits*, MS Thesis
Schuster, Mike

R:84 $5.00 *Design of the MOSAIC Processor*, MS Thesis
Lutz, Chris

M:84 $3.00 *Linguistic Analysis of Natural Language Communication with Computers*,
Thompson, Bozena H

R:84 $6.00 *Supermesh*, MS Thesis
Su, Wen-king

R:84 $14.00 *Mossim Simulation Engine Architecture and Design*,
Dally, Bill

R:84 $8.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

M:84 $3.00 *ASK As Window to the World*,
Thompson, Bozena, and Fred Thompson

R:83 $22.00 *Parallel Machines for Computer Graphics*, PhD Thesis
Ulner, Michael

M:83 $1.00 *Ray Tracing Parametric Patches*,
Kajiya, James T

'R:83 $9.00 *Graph Model and the Embedding of MOS Circuits*, MS Thesis
Ng, Tak-Kwong

'R:83 $2.00 *Stochastic Estimation of Channel Routing Track Demand*,
Ngai, John

'M:83 $2.00 *Residue Arithmetic and VLSI*,
Chiang, Chao-Lin and Lennart Johnsson

'R:83 $2.00 *Race Detection in MOS Circuits by Ternary Simulation*,
Bryant, Randal E

'R:83 $9.00 *Space-Time Algorithms: Semantics and Methodology*, PhD Thesis
Chen, Marina Chien-mei

'R:83 $10.00 *Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits*,
Lin, Tzu-Mu and Carver A Mead

'R:83 $4.00 *VLSI Combinator Reduction Engine*, MS Thesis
Athas, William C Jr

'R:83 $10.00 *Hardware Support for Advanced Data Management Systems*, PhD Thesis
Neches, Philip

'R:83 $4.00 *RTsim - A Register Transfer Simulator*, MS Thesis
Lam, Jimmy
current supply only: see *Acta Informatica* **20**, 301-313, (1983)

'R:83 $10.00 *Robust Sentence Analysis and Habitability*,
Trawick, David

'R:83 $12.00 *Automated Performance Optimization of Custom Integrated Circuits*, PhD Thesis
Trimberger, Steve

'R:82 $3.00 *Switch Level Model and Simulator for MOS Digital Systems*,
Bryant, Randal E

____5054:TM:82 $3.00 *Introducing ASK, A Simple Knowledgeable System*, Conf on App'l Natural Language Processing
Thompson, Bozena H and Frederick B Thompson

____5051:TM:82 $2.00 *Knowledgeable Contexts for User Interaction*, Proc Nat'l Computer Conference
Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho

____5035:TR:82 $9.00 *Type Inference in a Declarationless, Object-Oriented Language*, MS Thesis
Holstege, Eric

____5034:TR:82 $12.00 *Hybrid Processing*, PhD Thesis
Carroll, Chris

____5033:TR:82 $4.00 *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*,
Schuster, Mike, Randal Bryant and Doug Whiting

____5029:TM:82 $4.00 *POOH User's Manual*,
Whitney, Telle

____5018:TM:82 $2.00 *Filtering High Quality Text for Display on Raster Scan Devices*,
Kajiya, Jim and Mike Ullner

____5017:TM:82 $2.00 *Ray Tracing Parametric Patches*,
Kajiya, Jim

____5015:TR:82 $15.00 *VLSI Computational Structures Applied to Fingerprint Image Analysis*,
Megdal, Barry

____5014:TR:82 $15.00 *Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, PhD Thesis
Lang, Charles R Jr

____5012:TM:82 $2.00 *Switch-Level Modeling of MOS Digital Circuits*,
Bryant, Randal

____5000:TR:82 $6.00 *Self-Timed Chip Set for Multiprocessor Communication*, MS Thesis
Whiting, Douglas

____4684:TR:82 $3.00 *Characterization of Deadlock Free Resource Contentions*,
Chen, Marina, Martin Rem, and Ronald Graham

____4655:TR:81 $20.00 *Proc Second Caltech Conf on VLSI*,
Seitz, Charles, ed.

____3760:TR:80 $10.00 *Tree Machine: A Highly Concurrent Computing Environment*, PhD Thesis
Browning, Sally

____3759:TR:80 $10.00 *Homogeneous Machine*, PhD Thesis
Locanthi, Bart

____3710:TR:80 $10.00 *Understanding Hierarchical Design*, PhD Thesis
Rowson, James

____3340:TR:79 $26.00 *Proc. Caltech Conference on VLSI (1979)*,
Seitz, Charles, ed

____2276:TM:78 $12.00 *Language Processor and a Sample Language*,
Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

_____88-17	_____5238	_____5197	_____5135	_____5054
_____88-16	_____5236	_____5195	_____5134	_____5051
_____88-15	_____5235	_____5194	_____5133	_____5035
_____88-14	_____5234	_____5193	_____5132	_____5034
_____88-13	_____5233	_____5190	_____5129	_____5033
_____88-12	_____5232	_____5189	_____5128	_____5029
_____88-07	_____5231	_____5185	_____5125	_____5018
_____88-06	_____5230	_____5184	_____5123	_____5017
_____88-05	_____5229	_____5179	_____5122	_____5015
_____88-01	_____5228	_____5178	_____5114	_____5014
_____5258	_____5227	_____5174	_____5112	_____5012
_____5256	_____5223	_____5172	_____5106	_____5000
_____5253	_____5221	_____5168	_____5104	_____4684
_____5251	_____5220	_____5165	_____5094	_____4655
_____5250	_____5215	_____5164	_____5092	_____3760
_____5249	_____5214	_____5160	_____5091	_____3759
_____5247	_____5212	_____5158	_____5090	_____3710
_____5246	_____5210	_____5157	_____5089	_____3340
_____5244	_____5207	_____5147	_____5086	_____2276
_____5243	_____5205	_____5143	_____5082	
_____5242	_____5204	_____5140	_____5081	
_____5241	_____5202	_____5139	_____5074	
_____5240	_____5200	_____5137	_____5073	
_____5239	_____5198	_____5136	_____5065	

Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm

Wen-King Su and Charles L. Seitz
Department of Computer Science
California Institute of Technology

1. Introduction

We have been using variants of the Chandy-Misra-Bryant (CMB) distributed discrete-event simulation algorithm [1,2,3] since 1986 for a variety of simulation tasks [4]. The simulation programs run on multicomputers [5] (message-passing concurrent computers), such as the Cosmic Cube, Intel iPSC, and Ametek Series 2010. The excellent performance of these simulators led us to investigate a family of variants of the basic CMB algorithm, including lazy message-sending, demand-driven operation with backward demand messages, and adaptive adjustment of the parameters that control the laziness.

These studies were also motivated by our interest in scheduling strategies for reactive (message-driven) multiprocess programs [5,6,7], which are semantically similar to discrete-event (event-driven) simulators. The simulator itself is implemented in the reactive programming environment that we have developed for multicomputers, the Cosmic Environment, and the Reactive Kernel [8].

This paper is a brief and preliminary report of the simulation algorithms and performance results. A more definitive report will be found in the first author's forthcoming PhD thesis.

2. The CMB Simulation Framework

As usual, the system to be simulated is modeled as a set of communicating elements. A CMB simulator can be implemented by coding the behavior of elements in processes that communicate by messages. A message conveys both a time interval and any events within this interval. A process reacts to the receipt of an input message by updating its internal state; and, if outputs can be advanced in time,

The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

by sending messages to connected processes. These messages may include *null messages* that convey no events (changes in the state information), but serve only to advance the simulation time.

It is easy to show that such a simulator is correct [3], in the sense that it computes a possible behavior of the system being simulated. A sufficient condition for freedom from deadlock in this eager message-sending mode is that there is a positive delay in every circuit in the graph of element vertices and communication arcs. Intuitively, it is the delay of the elements being simulated that permits the element simulators to compute the outputs over an interval that is later than the time of the inputs, so that time advances. Simulation time is determined locally, and may get as far out of step at different elements as their causal relationships permit.

This conservative (also known as pessimistic) type of simulator exploits precisely the concurrency inherent in the system being simulated. In practice, just as with other concurrent programs, if the number of concurrently runnable processes substantially exceeds the number of processors, the utilization of concurrent resources is high. The speculative (also known as optimistic) type of simulator attempts to exploit additional concurrency by computing beyond the interval during which inputs are defined, at the risk of having to roll back if the speculations prove incorrect. Such approaches are attractive for simulating systems whose inherent concurrency is insufficient to keep concurrent resources busy, and in which speculations can be made with high confidence. Our studies have concentrated on conservative variants of the CMB algorithm.

The principal trouble with naive implementations of conservative CMB distributed simulation programs is a volume of null messages that may greatly exceed the number of event-containing messages. This difficultly is most evident when simulating systems with many short-delay circuits having relatively low levels of activity.

In practice, an element simulator may take as long to process a null message as an event-containing message, particularly with simple elements such as logic gates. In distributing the simulation, we seek to reduce the time required to complete the computation; however, we have an immediate problem if the element simulators must perform many more message-processing operations in the distributed simulation than they would perform event-processing operations in a sequential simulation. The centralized regulation of the advance of time achieved through the ordered event list maintained by sequential simulation programs allows these simulators to invoke element routines only once for each input event. The null messages inflate not only the volume of messages the system must handle, but also the computational load. Thus, if we are going to compete with the best sequential simulators, we must reduce the volume of null messages.

3. Indefinite Lazy Message Sending

To reduce the volume of messages, we use various strategies to defer sending outputs in the hope that the information can be packed into fewer messages. For example,

one of the most obvious schemes is to defer sending null messages, so that a series of null messages and an event-containing message can be combined to form a single message that spans a longer interval. Since output events are often triggered only by input events, deferring the delivery of preceeding null messages is less likely to hamper the progress of the destination element than deferring the delivery of event-containing messages.

The first problem that must be addressed in employing such strategies is deadlock. When element simulators defer sending output messages, they may cyclically deny themselves input messages, leading to deadlock. All of our simulators have employed a technique of *indefinite lazy message sending* to permit arbitrary strategies for deferring message sending, while still avoiding deadlock. The following is the inner loop of the simulator, shown in the C programming language:

```
while(1)
    if (p = xrecv())
        simulate_and_optionally_send_messages(p);
    else
        take_other_action();
```

The function xrecv returns a pointer, p, that points to a message for the simulation process if a message has been received. The simulator then dispatches to the appropriate element simulator, and may either send or queue the outputs that the element simulator produces. If there is no message in the node's receive queue, the pointer returned is a NULL (0) pointer. In this case, the simulator takes other action to break any possible deadlock. For a source-driven simulator, it selects a queued output to send as a message. For a demand-driven simulator, it selects a blocked element and sends a *demand* message to its predecessor to request that queued outputs be sent. A deadlock in deferring messages cannot occur without "starving" a node of messages. When this situation is detected by xrecv returning a NULL pointer, the resulting action breaks the potential deadlock.

Within this indefinite lazy message-sending framework, we can experiment with *any* scheme for deferring and combining messages without concern for deadlock. A message is free to carry any number of events, and an element is free to defer message sending on any basis.

4. Variant Algorithms

We have experimented with many CMB variants; in the interests of comprehension, we will outline the operation and report the performance of six variants that are representative of the range of possibilities that we have studied:

A Eager message sending: This basic form of CMB serves as a baseline for comparison against the variants.

B Eager events, lazy null messages: Null outputs are queued. Event outputs are sent immediately combined with any queued null outputs. When xrecv returns

a NULL pointer, the null output that extends to the earliest time is sent as a null message.

C ***Indefinite lazy, single event:*** All output from element simulators is queued. Messages are sent only when xrecv returns a NULL pointer. The output queue that extends to the earliest time is selected to generate a message up to the first event, if any, or a null message to the end of the interval.

D ***Indefinite lazy, multiple event:*** This scheme is a slight variation on *C*, motivated by characteristics of multicomputer message systems that make it economical to pack multiple events into fewer messages. All output from element simulators is queued. The output queues may contain multiple events. When xrecv returns a NULL pointer, the output queue that extends to the earliest time is selected to generate a message up to the *last* queued event, if any, or a null message to the end of the interval. However, to allow a direct comparison with sequential simulators, events are processed singly.

E ***Demand driven:*** Although we usually think of simulation as source driven from inputs, one can equally well organize the simulation as demand driven from outputs. In the pure demand-driven form, all output from element simulators is queued. When xsend returns a NULL pointer, the input that lags furthest behind selects the destination for a demand message. Upon receipt of a demand message, if the output queue is not empty, the simulator sends all the information in the output queue; if the output queue is empty, the simulator generates another demand message to the source of lagging input to this element.

F ***Demand driven adaptive:*** Demand messages single out critical paths in a simulation. In an adaptive form of demand-driven simulation, a threshold is associated with each communication path. Outputs of element simulators are queued only up to the threshold; when the threshold is exceeded, the contents of the queue are sent as a message. Demand messages operate as in *E*, but also cause the threshold to be decreased (in the cases shown below, the threshold is halved). The simulator is accordingly able to adapt itself to the characteristics of the system being simulated.

Although these variants are described here in terms of message passing, the same variants also appear as different scheduling strategies in shared-memory implementations.

5. Experimental Method

In common with other highly evolved message-passing programs, the simulator is implemented with one simulation process per multicomputer node (or, in the Cosmic Environment, with one simulation process per host computer or per processor in a multiprocessor). The instrumented simulator is actually a simulator within a simulator.

Basis of comparison: Although real-time execution speed is one of the most natural bases of comparison between any two programs that perform the same

function, real-time speed and speedup curves are not themselves particularly revealing when there are so many parameters involved.

In order to unmask the behavioral differences of the simulators, we normalize the measured execution speeds to a common unit, called a *sweep* [5, 6]. Here we will let a sweep be a fixed time required to process one message, whether a single event, null message, or demand message. The number of sweeps required for a sequential simulator to complete a simulation is simply the number of events generated during the simulation.

Instrumentation: The simulator is a reactive program written in C, and is instrumented to function in two operational modes. In the *emulation mode*, a multicomputer emulation program runs a simulation of a multicomputer; this in turn runs the reactive simulators. Speed is measured in sweep units. On each sweep, each node is allowed to get one message from its receive queue (if not empty) and process it. In the *real mode*, the simulator runs directly on the multicomputer. There is one copy of the simulator process in each node, and each simulator process runs a subset of the elements as embedded reactive processes. Each node runs at its own pace, and speed is measured with UNIX's real-time clock.

6. Experimental Results

We have performed these studies using logic circuits, because it is easy to construct examples with a diversity of behaviors, and because logic simulation is itself of practical interest. Performance measurements have been made on a variety of logic circuits, including those that are representative of circuits found in computers and VLSI chips, and those that are designed specifically to test or to stress the simulator. Six different network types, each in several sizes up to 4000 logic gates, have been the principal vehicles for these experiments. A larger range in performance is observed among circuits with different characteristics than between algorithm variants.

Multiplier example: The parallel multiplier is a good example of an ordinary logic circuit. It contains only limited concurrency: An n-bit multiplier has an average concurrency of $2n$ due to the sequential dependency in the paths for carry and sum. It does not contain tight loops that give the simulator artificial boosts or troubles, depending on element distribution and loop stability. It also contains moderately high fanout in the multiplier and multiplicand lines, which puts pressure on the message system. In all fairness, the distributed simulation of this multiplier circuit is not expected to do too badly or too well on a multicomputer.

For the simulation, the most-significant bit of the product is connected back to the multiplier input via an inverting delay. The delay is such that the multiplier reaches a stable state before the multiplier input changes. The multiplicand input is set to a value that causes the circuit to oscillate. A trace of the product outputs shows that the simulator and the circuit are running correctly.

Measurements in the emulation mode: In the emulation mode, a 14-bit multiplier is used. Each full adder is composed of seven logic gates, and the 14×14 structure contains a total of 1376 logic gates. The average number of concurrent events

is about 28. The plot in Figure 1 portrays in a log-log format the sweep count versus the number of nodes, N. The heavy horizontal line represents the number of sweeps a sequential simulator requires. The first remarkable characteristic of these performance measures is that they are so similar across this class of variant algorithms.

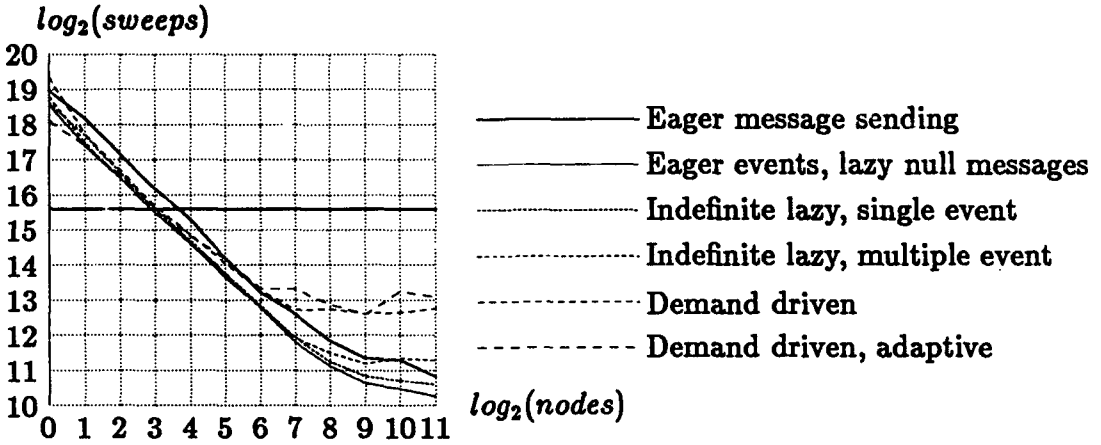


Fig 1: A 1376-gate multiplier, emulation mode

At $N=2^0=1$ node, we can compare the CMB variants with the sequential event-driven simulator. The concurrent simulators produce 4–10 times as many null or demand messages as event-containing messages, which is consistent with the 2–3 octave increase in sweep count over that of the sequential simulator. The speedup is close to linear in N for 5–8 octaves. The concurrent simulators do not become competitive with the sequential simulator until about $N=8$, but continue to nearly halve the sweep count with each doubling of resources until limiting effects are reached.

The demand-driven simulation modes E–F begin to perform poorly due to an increase in the volume of demand messages when the available concurrency of 28 ($\approx 2^5$) in the system being simulated is exhausted. In the adaptive form, demand messages are meant to make small-delay circuits more eager by reducing their queueing threshold. However, because the multiplier does not contain any small-delay circuits, demand messages drive the queueing threshold too low, and cause an excessive volume of null messages.

The source-driven variants extend the linear speedup for about 3 more octaves until the extra concurrency introduced by the null messages is also exhausted. These simulators reach asymptotic minimal time at 5 octaves below that of the sequential simulator, with only 3–6 elements per node. At this point the available concurrency is exhausted, and the number of elements per node is too small for the weak law of large numbers to assure load balance. The placement of elements in nodes for these trials is balanced but random.

Additional statistics have been collected to measure other effects. For example, when there are many circuit elements per node, the simulators are quite insensitive

to latency. When there are few elements per node, the performance begins to deterioriate as message latency is increased, particularly for the variants that perform well.

A second example for comparison: Figure 2 shows the sweep count versus N for a 3400-gate clock network. This asynchronous sequential circuit has many small-delay closed signal paths and a high activity level, resulting in an average event concurrency of 256.

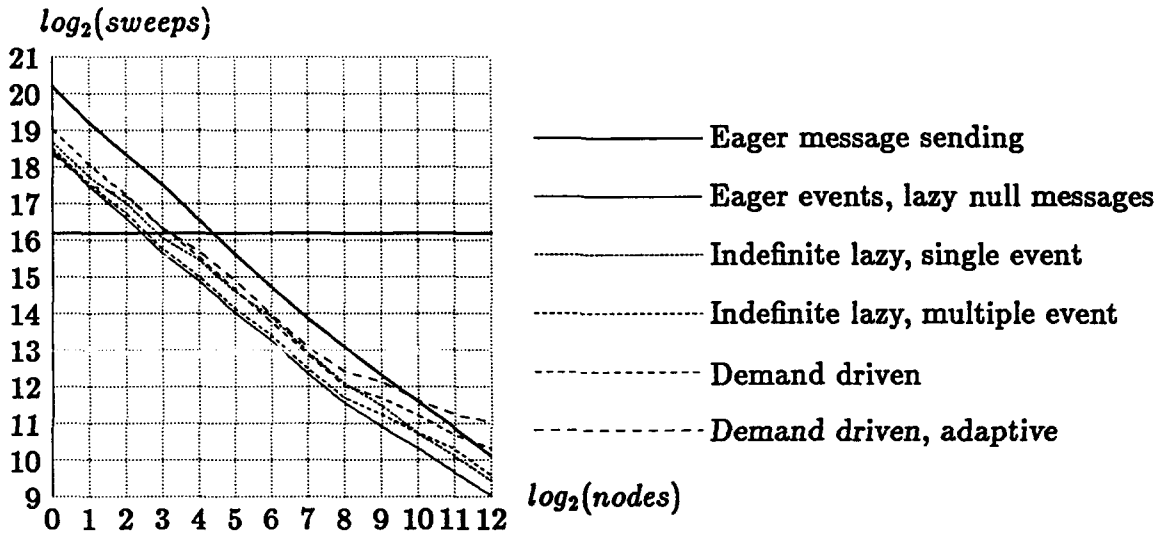


Fig 2: A 3400-gate clock network, emulation mode

Measurements on a real multicomputer: The results of simulating a scaled-down, 4-bit multiplier with 116 logic gates on an Intel iPSC/1 is shown in Figure 3. Simulation of larger circuits gives excellent but uninteresting results, with linear speedup over the entire range of $1 \leq N \leq 64$. (Due to limitations of the iPSC/1 message system, neither of the demand-driven simulation modes will run.) The timing results show that the reactive simulators require about twice as many calls to element simulators than a sequential simulator. The one-octave overhead is less than that of the 14-bit multiplier because a larger fraction of the elements are active. Since the average concurrency of the circuit is around eight, concurrency introduced by the circuit and by the null messages is expected to be exhausted when $N \geq 16$ nodes. Although the elapsed time plot shows that the time starts to level off when there are more than 16 nodes, it is somewhat less than linear in the range from 1–16 nodes, and is still decreasing slowly out to 64 nodes. The sublinear speedup is due to message latency in inter-node communications, increased null messages as the simulation is increasingly distributed, and load imbalance.

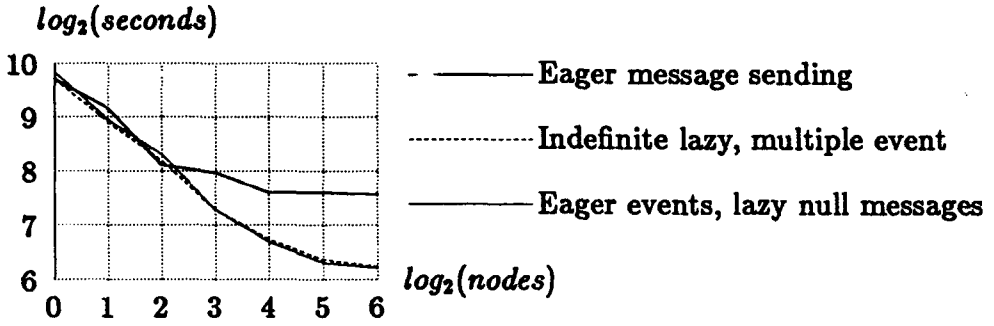


Fig 3: A 116-gate multiplier on an iPSC/1 for a 100μs period

7. Conclusions

Logic simulation, which involves simulating the behavior of relatively simple elements that have a high degree of connectivity, would be expected to be a difficult case for distributed simulation. Indeed, the simulations presented here have been much more revealing of the limitations of multicomputers and of the distributed discrete-event simulation algorithms than earlier simulations that we performed of systems such as multicomputer message networks.

For small N, neither the basic CMB algorithm nor the variants that we have tried are nearly as efficient for logic simulation as the sequential event-driven simulator. The null message is simply not as powerful a synchronization mechanism as the global ordered event list. However, for large logic circuits, these conservative variants on CMB produce excellent performance on multicomputers with large N and small message latency.

Our current efforts are to implement what we believe will be an entirely practical logic simulator for multicomputers and multiprocessors. It will employ a sequential event-driven simulator with an ordered event list in each node, and these simulators will be tied together using variants B, C, or D. Instead of random element placement, we will compute a placement that localizes small-delay circuits.

8. Acknowledgment

We very much appreciate the constructive suggestions, ideas, and encouragement that we have received from K. Mani Chandy.

9. References

[1] K. Mani Chandy and Jayadev Misra, "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations," *CACM* 24(4), pp 198–205, April 1981.

[2] Randal E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[3] Jayadev Misra, "Distributed Discrete-Event Simulation," *Computing Surveys* 18(1), pp 39–65, March 1986.

[4] "Submicron Systems Architecture," Semiannual reports to DARPA, Caltech Computer Science Technical Reports [5220:TR:86] and [5235:TR:86], 1986.

[5] William C. Athas and Charles L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer* 21(8), pp 9–24, August 1988.

[6] William C. Athas, "Fine Grain Concurrent Computation," Caltech Computer Science Technical Report (PhD thesis) [5242:TR:87], May 1987.

[7] William J. Dally, *A VLSI Architecture for Concurrent Data Structures,* Kluwer Academic Publishers, 1987.

[8] Charles L. Seitz, Jakov Seizovic, and Wen-King Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming," Caltech-CS-TR-88-1, January 1988.

Adaptive Routing in Multicomputer Networks

John Y. Ngai
Charles L. Seitz
California Institute of Technology*

Multicomputer Networks. Message-passing concurrent computers, more commonly known as *multicomputers*, such as the Caltech Cosmic Cube [1] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes [2]. The existing communication networks of the second-generation machines such as the Ametek 2010 employ an *oblivious* wormhole routing technique [6,7] which guarantees deadlock freedom. The message latency of these highly evolved oblivious technique have reached a limit of being as fast as physically possible while capable of delivering, under random traffic, a *stable* maximum substained throughput of $\approx$ 45 to 50% of the limit set by the network bisection bandwidth. Any further improvements on these networks will require an *adaptive* utilization of available network bandwidth to diffuse local congestions.

In an adaptive multi-path routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. It is expected that such an adaptive control can increase the throughput capability towards the bisection bandwidth limit, while maintaining a reasonable network latency. While the potential gain in throughput is at most only a factor of 2 under random traffic, the adaptive approach offers additional advantages such as the ability to diffuse local congestions in unbalanced traffic, and the potential to exploit inherent path redundancy in these richly connected networks to perform *fault-tolerant* routing. The rest of this paper consists of a brief outline of the various issues and results concerning the adaptive approach studied by the authors. A much more detailed exposition can be found in [3].

*The research described in this report was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597, and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

Adaptive Cut-through Routing. In any adaptive routing scheme which allows arbitrary multi-path routing, it is necessary to assure communication deadlock freedom. A very simple technique that is *independent* of network size and topology, is through voluntary *misrouting* as suggested in [4] for networks that employ data *exchange* operations, and more generally in store-and-forward networks. It was clear from the beginning that in order for the adaptive multi-path scheme to compete favorably with the existing oblivious wormhole technique, it must employ a switching technique akin to *virtual cut-through* [5]. In cut-through switching, and its blocking variant used in oblivious wormhole routing, a packet is forwarded immediately upon receiving enough header information to make a routing decision. The result is a dramatic reduction in the network latency over the conventional store-and-forward switching technique under light to moderate traffic. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly *matched*. A simple way is to have all *bidirectional* channels of the same node operate *coherently*. Observe that in the extreme, packets coming in can always be either forwarded or misrouted, even if the router has no internal buffer storage. In practice, buffers are needed to allow packets to be injected into the network, and to increase the performance of the adaptive control.

Network Progress Assurance. The adoption of voluntary misrouting renders communication deadlock a non-issue. However, misrouting also creates the burden to demonstrate progress in the form of message delivery assurance. An effective scheme that is independent of any particular network topology is to resolve channel access conflicts according to a *priority* assignment. A particularly simple priority scheme assigns higher priorities to packets that are closer to their destinations. Provided that each node has enough buffer storage, this priority assignment is sufficient to assure progress, *ie.*, delivery

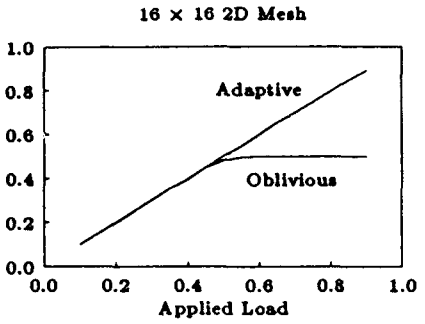


Figure 1: Throughput versus Applied Load.

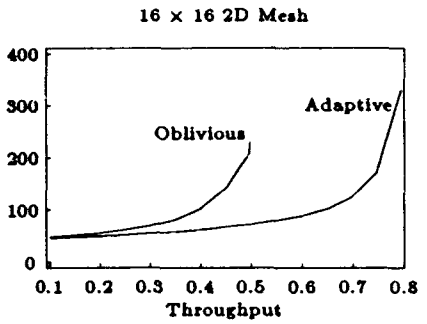


Figure 2: Message Latency versus Throughput.

of packets in the network. A more complex priority scheme that assures delivery of *every* packet can be obtained by augmenting the above simple scheme with *age* information, with higher priorities assigned to older packets. Empirical simulation results indicate that the simple distance assignment scheme is sufficient for almost all situations, except under extremely heavy applied load.

Fairness in Network Access. A different kind of progress assurance that requires demonstration under our adaptive formulation is the ability of a node to inject packets eventually. Because of the requirement to maintain strict balance of input and output data rates, a node located in the center of heavy traffic might be denied access to network indefinitely. One possible way to assure network access is to have each router set aside a fraction of its internal buffer storage exclusively for injection. Receivers of packets are then required to return the packets back to the senders, which in turn reclaim the private buffers enabling further injections. In essence, the private buffers act as *permits* to inject, which unfortunately have to be returned back to the original senders, thereby wasting network bandwidth. A different scheme that does not incur this overhead is to have the nodes maintain a bounded synchrony with neighbors on the total number of injections. Nodes that fall behind will, in effect, prohibit others from injecting until they catch up. With idle nodes handled appropriately, the imposed synchrony assures eventual network access at each node having packets queued for injection.

Performance Comparisons. An extensive set of simulations were conducted to obtain information concerning the potential gain in performance by switching from the oblivious wormhole to the adaptive cut-through technique. Among the various statistics collected, the two most important performance metrics in communication networks are *network throughput* and *message latency*. Figure 1 plots the substained normalized network throughput versus the normalized applied load of the oblivious and adaptive schemes for a 16 × 16 2D mesh network, under random traffic. The normalization is performed with respect to the network bisection bandwidth limit. Starting at very low applied load, the throughput curves of both schemes rise along a unit slope line. The oblivious wormhole curve levels off at ≈ 45 to 50% of normalized throughput but remains *stable* even under increasingly heavy applied load. In contrast, the adaptive cut-through curve keeps rising along the unit slope line until it is out of the range of collected data. It should be pointed out, however, that the increase in throughput obtained is also partly due to the extra silicon area invested in buffer storage, which makes available adaptive choices. Figure 2 plots the message latency versus normalized throughput for the same 2D mesh network for a typical message length of 32 flits. The curves shown are typical of latency curves obtained in virtual cut-through switching. Both curves start with latency values close to the ideal at very low throughput, and remain relatively flat until they hit their respective transition points, after which both rise rapidly. The transition points are ≈ 40% and 70%, respectively for the oblivious and adaptive schemes. In essence, the adaptive routing control increases the quantity of routing service, *ie.*, the network throughput, without sacrificing the quality of the provided service, *ie.*, the message latency, at the expense of requiring more silicon area.

Fault-tolerant Routing. Another area where adaptive multi-path routing holds promise is in fault-tolerant routing. The opportunity here stems from the fact that, as we continue to build larger machines, we expect faults to be increasingly probable. However, for performance reasons, the networks popular in multicomputers are already very rich in connectivity. It is conceivable that a multi-

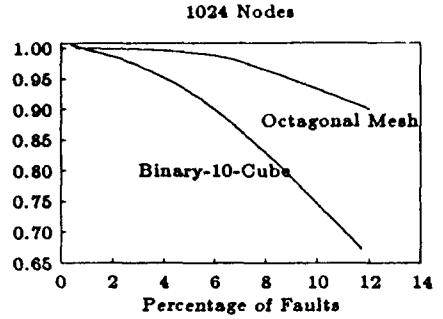


Figure 3: Reclamation Ratio for Node Faults

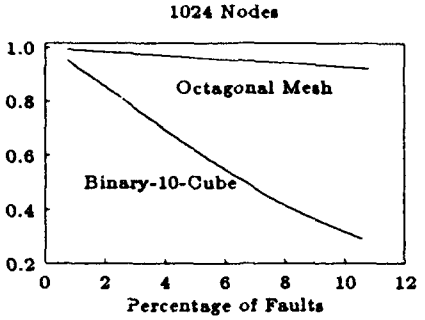


Figure 4: Reclamation Ratio for Edge Faults

path control can perform fault-tolerant routing simply by exploiting the *inherent path redundancy* in these networks. Fault-tolerant routing has been intensively studied in the network research community. However, multicomputer networks impose stringent restrictions, not present in traditional networks, that require a new approach. In particular, observe that the popular connection topologies of multicomputer networks such as k-ary n-cubes or meshs are highly *regular*, which allow for simple algorithmic routing procedures based entirely on local information. Such capability is particularly important in fine-grain multicomputers where resources at each node are scarce. Equally important, the simple algorithmic routing procedures in these regular topologies allow direct hardware realization of the routing functions, which is absolutely essential in high performance systems.

As nodes and channels fail, the regularity of these networks is destroyed and the algorithmic routing procedures are no longer applicable. Routing in irregular networks can be achieved by storing and consulting routing tables at each node of the network. However, such a scheme demands excessive resources at each node and becomes unacceptable as the networks grow in size. A different and more satisfactory approach exploits the regularity of the original non-faulty network. An interesting example of such an approach can be found in [8]. In this paper, we suggest an alternate approach based on our adaptive routing formulation. Instead of devising ways to route messages in these semi-irregular networks, we seek ways to *restore* the original regularity of the survival networks. This approach allows us to continue to use the original algorithmic routing procedure. One immediate advantage is that the faulty network can continue to use the original hardware router with very little change. Another advantage of this approach is that we can obtain *a priori* bounds on the length of routes joining pairs of sources and destinations in the faulty network.

Regularization Procedures. An immediate result of having only local information to guide routing is that, pairs of survived nodes may not be able to communicate with each other even if they remain connected. In order to communicate, each pair must have at least one unbroken *route* joining them, which belongs to the set of original routes generated algorithmically in the non-faulty network. Because of its resemblance to the notion of *convexity*, we refer to them as *convex networks*. Starting with an irregular survived network, one way to restore regularity is to selectively *discard* a subset of the survived nodes, so that the remaining subset becomes *convex*, and hence can still communicate with each other according to the original algorithmic procedure. In essence, nodes which become difficult to reach without global information are abandoned as a result of our insistence on using only local routing information. Another technique that can be employed to restore regularity is to selectively *restrain* a subset of the survived nodes to operate purely as routing switches, *ie.*, they are not allowed to source or consume messages. The rationale is that some survived nodes which are difficult to reach from everywhere, and hence should be discarded, may be in positions which enable other pairs to communicate, and hence should be retained.

Some Reclamation Results. It is clear that the effectiveness of this regularization approach will ultimately depend on the connection topology and the routing relations defined by the algorithmic routing procedure. High-dimensional networks such as the binary n-cube are expected to deliver good results, whereas low-dimensional ones such as the 2D meshes generally do not. One possible way to improve the reclamation yield of these low-dimensional networks is to *augment* them with extra channels, *eg.*, adding diagonal connected channels to a 2D mesh results in

an *octagonal mesh*. The additional connectivity in the octagonal mesh generates a much richer set of paths, and hence delivers much better reclamation yield. Figures 3 and 4 plot the reclamation ratio for the 32×32 octagonal mesh and Binary-10-cube versus the fraction of node faults, and channel faults respectively. The faults were generated independently and uniformly over the specific networks.

Future Challenge. Many aspects and problems have been addressed in the course of this research, and a number of solutions have been found. Clearly, more work remains to be done. Perhaps the most challenging of all is to realize on *silicon*, the set of ideas outlined in this study.

References.

[1] Charles L. Seitz, "The Cosmic Cube", *CACM*, 28(1), January 1985, pp. 22-33.

[2] William C. Athas, Charles L. Seitz., "Multicomputers: Message-Passing Concurrent Computers", *IEEE Computer*, August 1988, pp. 9-24.

[3] John Y. Ngai, *Adaptive Routing in Multicomputer Networks*. Ph.D. Thesis, Computer Science Department, Caltech. To be published.

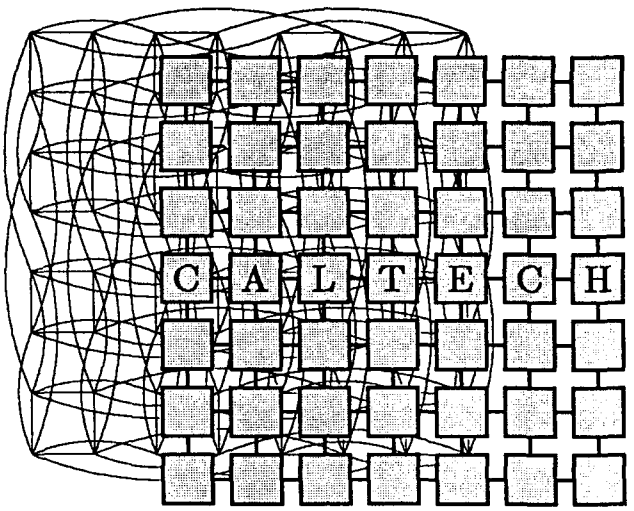
[4] Borodin, A. and Hopcroft, J., "Routing, Merging, and Sorting on Parallel Models of Computation", *Journal of Computer and System Sciences*, **30**, pp. 130-145 (1985).

[5] P. Kermani and L. Kleinrock, "Virtual Cut-Through : A New Computer Communication Switching Technique", *Computer Networks* 3(4) pp. 267-286, Sept. 1979.

[6] William J. Dally and Charles L. Seitz, "The torus routing chip", *Distributed Computing*, 1986(1), pp. 187-196.

[7] Charles M. Flaig, *VLSI Mesh Routing Systems*. Caltech Computer Science Department Technical Report, 5241:TR:87.

[8] J. Hastad, T. Leighton, M. Newman, "Reconfiguring a Hypercube in the Presense of Faults." *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. May, 1987.



SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-88-5

7 April 1988

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-5

7 April 1988

Reporting Period: 1 November 1987 – 31 March 1988

Principal Investigator: Charles L. Seitz

Faculty Investigators: William C. Athas
K. Mani Chandy
Alain J. Martin
Martin Rem
Charles L. Seitz

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of the research activities and results for the five-month period, 1 November 1987 to 31 March 1988, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

Some highlights of the previous five months are:

- The Ametek Series 2010, a second-generation medium-grain multicomputer developed as a joint project between our research project and Ametek Computer Research Division, was announced as a commercial product. A 16-node engineering prototype has been demonstrated running numerous application programs. (See section 2.1 and the paper "The Architecture and Programming of the Ametek Series 2010 Multicomputer" in the appendix.)
- Enhancements to the Cantor programming system (section 3.1).
- Reference definition of the functions of the Cosmic Environment and Reactive Kernel (sections 3.2 and 3.3).
- High-quality self-timed VLSI designs are being produced by a compilation procedure that is now fully automatic (sections 4.1 and 4.2).
- Fast "Mesh Routing Chips" (section 4.5).

2. Architecture Experiments

2.1 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Alain Martin, Bill Athas, Charles Flaig, Jakov Seizovic, Craig Steele, Wen-King Su

On 19 January 1988, the Ametek Series 2010 multicomputer was announced at the 1988 Hypercube Conference in an invited talk by Chuck Seitz. This is the first multicomputer to reach our goal for the second generation of multicomputers of a 100× improvement over the first-generation hypercube multicomputers in the relationship between communication and computing performance. A paper on "The Architecture and Programming of the Ametek Series 2010 Multicomputer," to appear in the proceedings of the 1988 Hypercube Conference, is included as an appendix to this report.

In this same week, a 16-node engineering prototype of the Ametek Series 2010 was demonstrated and benchmarked running application programs. All of these programs had been developed and run previously on Cosmic Cubes, Intel iPSC/1s, or "ghost cubes." In all cases, the programs ran correctly on the Ametek Series 2010, requiring only compilation and linking with the appropriate compatibility libraries. In March 1988, a 16-node system with 20 Mflop vector floating-point accelerators on each node was demonstrated running an edge-detection benchmark at 170 Mflops. Systems at the centerline design point of $N = 256$ nodes will be capable of a peak performance of 1 GIPS, 5 Gflops, and 5 Gb/s network bilateral bisection bandwidth.

The announcement and demonstration of the Ametek Series 2010 was the culmination of a 16-month joint development program with Ametek Computer Research Division. Our Caltech project provided the architectural design, routing chip designs and prototypes, and system software consisting of the Reactive Kernel (RK) node operating system and the Cosmic Environment (CE) host runtime system. Ametek provided the detail logical designs, physical designs, parts, assembly, and construction of the prototypes to our specifications and designs. Ametek also ported RK, and wrote the necessary interface routines to CE.

Considering the complexity of this project (new architecture, new system software, new custom mesh routing chips, new node design, new host interface, and new packaging), it proceeded very smoothly. The RK port required only two months for the Ametek system-programming team, and about 90% of the resulting system is identical to C source code provided by Caltech. The only serious problem that occurred in the entire project was routing chips that did not function correctly

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Ametek Computer Research Division (Monrovia, California).

on first silicon. This problem was traced to a missing contact cut and mistake in the signal naming that did not allow this error to be detected in the usual extraction and switch-level simulation process. The second-pass silicon on this self-timed SCMOS chip, one of two independent mesh routing chip designs, functioned correctly. The other design worked correctly on first silicon.

Ametek has non-exclusive licenses to Caltech patents on the Cosmic Cube architecture and message-passing mechanisms, to Caltech patents on mesh routing chip organization, and for Caltech system software. As part of this license arrangement, Ametek will be contributing a 256-node system to Caltech. An allocation of cycles on this system will be made available to guest researchers, as is currently done with our Cosmic Cubes and iPSC/1.

2.2 Mosaic Project

Bill Athas, Charles Flaig, Glenn Lewis, Don Speck, Wen-King Su, Chuck Seitz

The Mosaic C is a message-passing MIMD multicomputer with single-chip nodes. The stipulation that the nodes fit on a single chip limits the storage for each node, so that relatively fine-grain concurrent programming techniques must be used. We are working toward building a 16K-node Mosaic system using nodes fabricated in 1.2μm CMOS technology, with a near-term milestone of a 1K-node system using nodes fabricated in 2μm CMOS.

The status of the Mosaic C chip design is described in section 4.4, and the current work on the Cantor programming system that we shall use for programming the Mosaic is described in section 3.1.

2.3 Cosmic Cube Project

Bill Athas, Michael Lichter, Wen-King Su, Jakov Seizovic, Chuck Seitz

This section summarizes the current usage and the hardware and software status of our first-generation multicomputers, the Cosmic Cubes and Intel iPSC/1 d7. These systems continue to operate reliably. The major system software changes introduced in the fall 1987 have caused no significant problems, and have improved the compatibility between the Cosmic Cubes, iPSC/1, "ghost cubes," and the Ametek Series 2010.

Overall usage has been moderately heavy. The most time-consuming application in this period from within our own group have been an extensive series of simulations by John Ngai concerned with the maximal utilization of networks with faulty routers or channels (see section 4.6). Supersonic flow computations being performed by students and faculty in Aeronautics at Caltech continue as the largest share of outside use. Other guest users include David Mizell's group at ISI, who have been experimenting with distributed simulations, and several researchers doing neural network simulations.

Neither the 64-node nor 8-node Cosmic Cubes has exhibited a hard failure in this five-month period. These cubes have now logged 3.2 million node-hours with only three hard failures. The calculated node MTBF of 100,000 hours reported before these machines were constructed was extremely conservative. A node MTBF in excess of 1,000,000 hours is probable, and can be stated at a 50% confidence level.

Our Intel iPSC/1 d7 (128 nodes) was contributed to the Submicron Systems Architecture Project as a part of the license agreement between the Caltech and Intel, and is accessible via the ARPAnet to other DARPA researchers who may wish to experiment with it. To request an account, please contact `chuck@vlsi.caltech.edu`. Delivery of the alpha test unit of the new Ametek Series 2010 system is anticipated in about two months. This system will be available for outside use on a similar basis.

3. Concurrent Computation

3.1 Cantor

Bill Athas, Nanette Jackson, Chuck Seitz

Continuing research using the Cantor programming system has focused on writing application programs, and on refining the Cantor programming model. Our goal in writing application programs is to develop programs that are suitable for execution upon fine-grain multicomputers, such as the Mosaic C. Our experience from writing programs in Cantor is used to refine the Cantor language definition, and the instrumentation of these programs has provided the essential parameters for the design both of the Mosaic C and of an experimental Cantor Engine.

New applications programs written in Cantor include a program to enumerate paraffin isomer molecules, a program to test for graph isomorphism, and a program to analyze a chessboard for a checkmate configuration and report the possible moves to escape checkmate. This latter program is over 750 lines of code.

From these programs, plus the programs previously reported, we have observed three general paradigms for writing concurrent programs in Cantor.

1. The first paradigm is the transformation of functional or dataflow programs into Cantor programs. The transformations applied are systematic and the application of continuations and futures is straightforward.

2. The second paradigm is the transformation of a program specification into a Cantor program. The typical problems from this area are combinatorial searches using a breadth-first or divide-and-conquer approach. However, the resulting Cantor object graphs are not trees but are series-parallel (S-P) graphs. The S-P graphs are formed from the factoring of recursions into two parts: the invocation of the recursive call, and the rendezvous with the return from the recursion.

3. The third paradigm, and by far the most interesting, is the object program as an apparatus for performing a computation. A simple example is the wheel-driven prime sieve, in which the computation is represented by a number generator called the wheel and the infinite sieve. More interesting examples are simulation in which each object in the simulation is represented by a Cantor object.

Our latest revision of Cantor is version 2.2. This version supports dynamically-allocated vectors and functional abstraction. Cantor 2.1 supported vectors in which the size of the vector was computed at compile time. This restriction supported efficient compilation of vectors, but was of limited usefulness. We often found that vectors are combined into larger vectors, in which the size of the component vectors is data-dependent. Thus, Cantor 2.2 supports vectors that are allocated on demand.

Cantor 2.2 also provides for functional abstraction over expressions. Previously, functional application was used to produce future reference values for new objects. Functions can now produce a value of any type. The invocation of a function is quite similar to creating a new object. The function is expected to produce a list of return values for the caller. The list of return values is passed back to the caller by message-passing. In the interim between calling a function and receiving the list of return values, the caller leaves the running state. All messages received between calling a function and receiving the reply message are enqueued. Once the reply message returns, the object is again a candidate for execution, and all messages that were enqueued are processed using the normal execution rules. Because calling a function causes the caller to leave the running state, the context for the caller must first be saved. The saving of context is performed by the compiler using live-variable analysis.

Our next refinement for the Cantor programming system is to provide a facility for supporting custom objects and functions, namely, machine code that has been separately prepared, but which is compatible with the Cantor execution model. Our plan for incorporating custom objects and functions into Cantor are to provide for separate compilation of Cantor object definitions and functions, and then link the native definitions with the definitions for custom objects and custom functions.

The latest stable and distributed version of Cantor is 2.0. It is expected that version 2.2 will become available for distribution to other research groups in mid-summer.

3.2 The Cosmic Environment

Wen-King Su, Chuck Seitz

The Cosmic Environment (CE), our generic, portable multicomputer interface, has been augmented with the Unix standard IO libraries. This new feature was made possible by the addition of RPC messages. A RPC message is identical to a normal message, with the exception that a program has the option of selectively waiting for a reply message. When a program issues an `xrecvrpc`, the program is blocked until a RPC message is received. The message is then returned to the process.

The "ghost cube," a multicomputer simulator that is made of a group of NFS-connected UNIX computers or workstations, has proved to be very popular. Ghost cubes now have a hook for running the debugger program. Users can run dbx on their node programs and test their programs fully on a ghost cube before moving them unmodified to a real multicomputer. The Cosmic Environment now supports the original Cosmic Cubes, the Intel iPSC/1, ghost cubes, and the new Ametek Series 2010.

The documentation for CE version 7.2 and for the Reactive Kernel is now completely up-to-date. The latest edition of "The C Programmer's Abbreviated

Guide to Multicomputer Programming" (Caltech-CS-TR-88-1) was completed in January 1988, and 300 copies were distributed to our user community. CE has now been distributed to well over 100 sites in the United States, Canada, Western Europe, Scandinavia, and Israel.

3.3 The Reactive Kernel

Jakov Seizovic, Chuck Seitz

The Reactive Kernel (RK) has been successfully ported to one of the second-generation machines, the Ametek 2010, and has been running reliably on that system for the past three months. This uneventful port has demonstrated that our goal of making RK highly portable was achieved. The careful layering of the RK structure, with well-defined interfaces between the layers, has enabled the testing and tuning RK by incrementally adding more complex features, without interfering with the already tested ones. Much of this activity has been concerned with trying to get as much performance as possible out of the message system. The following back-reference problem is an example of this kind of tuning.

Consider the following program fragment, which occurs frequently in programs with the reactive primitives:

```
p = xmalloc (length);
build_the_message (p);
xsend (p,node,pid);
```

At the allocation time, a data structure called a *descriptor*, which contains the relevant information about the allocated block, is associated with the block. This scheme creates a *back-reference* problem; that is, a problem of finding the appropriate descriptor given the pointer to a particular memory block.

An obvious solution is to keep the descriptor pointer, or the whole descriptor, within the memory block itself. However, this solution is not satisfactory, because misuse or overwriting these pointers or descriptor by user processes can cause an operating system error. What we need is a dictionary, a set representation with the *insert*, *delete*, and *member* operations. The set elements are descriptors, and the *keys* are pointers to the memory blocks. The algorithm used in RK makes a compromise between the time and space complexity. The idea of the algorithm is as follows: in order to access an element of the set, we perform a search along an N-ary tree for k steps, whereby with each step we reduce the number of possible elements by a factor of N. After k steps we are left with at most $n = N_{max} N^k$ possible outcomes, and can resolve the remaining ambiguity, if any, by a sequential search.

Given the size of the memory used for messages, the average number of messages in the memory, the distribution of message sizes, and the cost function representing the balance between the memory utilization and the time required to access an

element of the set, we are able to find an optimal configuration. Since the parts of the data structure are dynamically allocated, it is even possible to change the configuration 'on-the-fly,' after obtaining the information about the current message traffic. If the reconfiguration is performed at the point when there are no messages in the system, it can be done with essentially zero cost.

The only important addition to RK functions that we are planning is a variant of the standard spawn function that places a process automatically. Associated with this addition will be an improved mechanism to cache process code, so that the speed of spawning a new process will be comparable to that of message passing. This addition is part of our long-term plan to make the semantics of a subset of RK message and spawning functions identical to those of Cantor (section 3.1).

3.4 Concise — A Concurrent Circuit Simulator*

Sven Mattisson, Lena Peterson, Chuck Seitz

The concurrent circuit simulation program, Concise, currently runs under the Cosmic Environment with the reactive primitives on UNIX computers; on all forms of multicomputers, including ghost cubes; and also on a Sequent under the Cosmic Environment.

Experimental modifications have been made over the past several months in order to make clustering of tightly coupled circuit nodes possible. The clustered "difficult" nodes are solved by a direct method, thus increasing the convergence rate for many circuits, both digital and analog. An investigation of automatic circuit partitioning methods is currently underway.

In another effort, Concise has been used by Anthony Skjellum in the Chemical Engineering Department at Caltech for the simulation of distillation columns. This work has shown that it is possible to use Concise to simulate dynamic systems that are not at all like circuits. As part of this effort, Concise has been modified to make it easier to install models of other kinds of "devices."

This work on Concise will be presented in two papers at the IEEE International Symposium on Circuits and Systems (ISCAS) in Helsinki, June 1988.

3.5 Transformational Derivation of Distributed Algorithms

Kevin S. Van Horn, Alain Martin

In the past several months we have begun to develop a transformational method for deriving concurrent programs, with an emphasis on the derivation of distributed programs. A transformational derivation of a concurrent program proceeds as

* This segment of our research is a joint project between the Caltech Submicron Systems Architecture Project and the Department of Applied Electronics at the University of Lund, Sweden.

follows. Given a problem to solve, one first produces a simple, easily-understood program with a straightforward correctness proof. This program may be inefficient, involve globally shared variables, make no use of message-passing, and may not even have any explicit concurrency. One then applies a series of transformations to this program, proving any conditions which must hold for the transformation to be valid, until one obtains an efficient distributed program.

There are several advantages to such a method. One is that the conceptual structure of the algorithm becomes much clearer. The original program expresses the essence of the algorithm, which is elaborated by succeeding transformations that, for example, implement global tests and updates of global variables, and detect termination. Another advantage is that the correctness proof of the final program is broken into smaller, more easily managed pieces. Perhaps the biggest advantage is that it allows one to work out and prove correct an intermediate solution to the problem before deciding on many details of the final algorithm.

The notation used is a variant of Chandy and Misra's UNITY. We are at present restricting ourselves to terminating programs in order to avoid some thorny issues that arise with non-terminating programs, although it appears that many of the transformation techniques developed so far should be applicable to both. A program in this notation consists of a declaration of variables with their initial values, a set of assignments, a termination condition, and a result expression. The operation of such a program can be described informally as follows: repeatedly (and fairly) choose an assignment or the termination condition; if an assignment is chosen then execute it, otherwise evaluate the termination condition and if it holds then terminate, returning the value of the result expression in the present state. The kinds of transformations we apply to these programs include data refinement, distributing and/or combining assignments, superposing new variables, removing superfluous variables, and strengthening the termination condition.

This transformational method has been used to derive a number of algorithms, some original and some preexisting. These include a distributed best-first search algorithm, various all-points shortest path algorithms, two termination-detection algorithms, and a distributed minimal spanning tree algorithm that appears to be a significant improvement over that of Gallager *et al.*

3.6 A Multicomputer Z–Buffer Program

Glenn M. Lewis, Wen-King Su, Chuck Seitz

As a demonstration program for multicomputers running the Reactive Kernel, we have written a distributed version of the usual graphics Z–buffer program. It takes input from any graphics rendering program that generates three–dimensional coordinates and color, and sorts the information such that the result simulates a true hidden-line representation of the image.

4. VLSI Design

4.1 Standard-cell Placement and Routing Program

Steve Burns, Pieter Hazewindus, Alain Martin

To facilitate rapid layout of chips, we have designed a new placement and routing program, `gladys`. This program takes as input a circuit description consisting of a set of gates, which may be generated by the circuit compiler. This description is then converted into a standard-cell layout. The result is a number of *towers* of standard cells, with routing channels in between towers. In the standard cells, no metal2 is used, so that the router can route between towers over standard cells.

The program consists of a placement algorithm, which attempts to reduce wire lengths by simulated annealing. Thereafter, global routing is done to route between cells in non-adjacent towers, and finally, a channel router does local routing in the channels between towers, using a greedy three-layer routing algorithm. The router has no global considerations when deciding on the location of wires; hence, the algorithm is very fast (it typically routes a medium-sized chip in a matter of seconds).

We have compared this algorithm with layouts generated by MOSIS's FUSION tool. The FUSION layout is about 50% larger if no placement is specified, and about 10% larger if it is supplied with the result generated by the previously mentioned placement algorithm. We expect to be able to reduce our layout size by 5–10% by using a better channel routing scheme, and by incorporating some global optimizations.

As a final step in the automatic transformation of a program into a chip, a padrouter needs to be constructed.

4.2 Bit-serial Routing Chip Compiled from a High-level Description

Steve Burns, Alain Martin

We have designed and fabricated a self-timed bit-serial routing chip compiled directly from a program. All stages of the compilation were performed automatically, using a procedure with the following structure:

(i) parse tree generation,

(ii) tree-based (global) optimization,

(iii) operator generation,

(iv) peephole (local) optimization,

(v) operator to standard-cell binding,

(vi) standard-cell placement, and

(vii) inter-cell routing.

Stages (i) through (v) were performed using a PROLOG-based 'CSP to Self-timed Circuit' compiler hinted at in the last semiannual DARPA report, and described in more detail at the 1988 MIT VLSI Conference. The placement and routing steps were performed by the MOSIS FUSION system.

The "Compiled MRC" was tested and functions correctly with a throughput of 5.6 MHz (four-phase handshake in 180 ns). The latency through a single router element is 253 ns. The performance of this chip is somewhat disappointing, caused mostly by an inadequate implementation of step (v). A more careful implementation of the 'operator to standard-cell binding' step should increase the performance of the compiled chips by a factor of two.

Global optimizations will also improve the circuits produced by this compilation method. In particular, reshuffling of communication actions will, in many cases, produce more efficient (in terms of area and speed) implementations. However, in general, reshuffling introduces deadlock. Global analysis of the system is necessary to show that reshuffling will not introduce deadlock. Currently, this global analysis is performed manually, and annotations are added to the source programs specifying when the communications may be interleaved. We are working to automate this analysis. The "Compiled MRC" included a router element with reshuffled communications. The throughput of the reshuffled router increased 20% to 6.7 MHz (four-phase handshake in 148 ns). The latency was reduced more dramatically to 81 ns.

4.3 Characterization of Communication Patterns with Constant Response Time

Tony Lee, Alain Martin

In a system of identical communicating processes connected in a regular structure (linear array or mesh) —such systems are usually called systolic arrays—, the order of communications of a process with its neighbors can be modified to improve performance. However, it is, in general, difficult to predict the effect of such a reordering: it may cause deadlock, or it may lead to a behavior where the "response-time" of a process to a communication depends on the number of processes in the systems.

It so happens that the reshuffling of actions in a handshaking expansion that we perform during the compilation of a communicating process into self-timed circuits have the same properties: although they are introduced to improve preformance, they may lead to deadlock or to a variable response-time.

We have defined a necessary and sufficient condition for a communication pattern in a linear array to be deadlock-free and to have a constant response-time.

4.4 Mosaic Elements

Bill Athas, Charles Flaig, Glenn Lewis, Don Speck, Wen-King Su, Chuck Seitz

The Mosaic C chip is composed of three main parts: RAM & ROM, channels, and processor. Our strategy for verifying the design of this very complex chip and characterizing its yield on MOSIS runs is initially to fabricate and test the three main parts separately. After the parts have been well characterized, their layouts will be combined onto a single chip. All the sections except for the ROM have been designed and layed out. The RAM and channels sections have been fabricated and verified. The final assembly of the processor and of the entire chip are expected to be accomplished this summer.

The target technology for the Mosaic C is MOSIS SCMOS with $0.6\mu m \leq \lambda \leq 1.5\mu m$. Target maximum chip size is $36mm^2$, or $100M\lambda^2$ with $\lambda = 0.6\mu m$, and $16M\lambda^2$ with $\lambda = 1.5\mu m$. Speed, storage size, and top-level floorplan will necessarily vary with feature size.

The architecture of the Mosaic C and the design of the Mosaic C chip are described in previous semiannual technical reports.

4.4.1 Mosaic C dRAM

Our basic strategy has been to develop a 4-transistor dRAM that is a low-risk design with a relatively large area, and a 2-transistor dRAM that is a higher-risk design but has a relatively small area. The following efforts have been aimed at improvements in the 4T dRAM:

Decoders: Due to pitch constraints, the RAM and ROM row select decoders must be precharged. Our desire to charge and discharge as few decoder outputs as possible leads us to domino NAND gates.

However, precharging through a series transistor chain can be very slow. Because the transistors are turning off as charge is drained, the precharge time (and hence the input setup time) is cubic in the chain length. The setup time allowance for the decoder is zero, so each internal node must have its own precharger. To make room for those prechargers, series chains must be coalesced into trees, with a branching width limited to 70λ so that internal nodes remain accessible and stay small enough to not need area-consuming metal strapping.

For speed it's conventional to predecode bit pairs so that fewer series transistors are needed. However, with the tapering transistor sizes afforded by the tree structure, the time saved by removing half of the transistors does not recoup the predecode overhead. Predecoding only gains speed if applied just to the leaves of the trees, where the predecode time is not in the critical path.

RAM simulation: We have discovered a bug in SPICE2G.6 which greatly overestimates the effective gate capacitance of pass transistors. When a pass

transistor is cut off by back-gate bias, the CMEYER routine calculates full gate capacitance, as if the MOS capacitor were in accumulation, when it should be in deep depletion with a much lower capacitance.

SPICE2G.6 also neglects the channel-to-bulk capacitance, though that bug at least has an easy workaround (increase the source area by the amount of gate area).

4.4.2 Channels

The width of the channels in our current designs has been increased to 4 bits. The registers and bus drivers for the processor interface have been completed, and state tables for the control circuitry have gone through a first draft.

4.4.3 Processor

The Mosaic C processor datapath design and layout is complete, and it simulates correctly with MOSSIM. Our efforts of the past several months have included continued checking of the microcode, and attempts to improve the speed of the control PLA.

4.5 Self-Timed Mesh Routing Chips

Charles Flaig, Chuck Seitz

Samples of the Mesh Routing Chips (MRCs) sent to MOSIS for fabrication in September were received and tested in December, and functioned correctly. The 95% yield was excellent, but the speed was below expectations. The cycle times for these chips was about 100ns in 3μm SCMOS, which was a factor of two less than expected. The fallthrough time for each FIFO stage was also high, at about 15-20ns. A large part of the problem was traced to long wires in the 2/4-cycle conversion circuitry. A design oversight placed excessive capacitive loads on relatively weak transistors. There were also some "hurry up" design shortcuts that were detrimental to the speed. Based on experience with another MRC design, this design would have exhibited a satisfactory cycle time of about 33ns in 1.6μm SCMOS, but our studies of the internal timing of this chip revealed a way to increase the speed quite dramatically.

A new version of the MRC was begun in December. This design corrects all of the known problems and shortcuts in the original MRC, but also implements the FIFOs and internal switching with a more efficient signaling scheme. The external signaling conventions must still conform to the MRC specification. The major internal changes are as follows:

1. New FIFO stages. The previous FIFOs used interconnected C-elements which would store a flit (flow control unit) in two successive stages. The new FIFOs use some additional state and timing information to produce a load pulse of fixed

width, and thus store a flit in a single stage. While this does not significantly affect the fallthrough or cycle time, it increases the amount of storage available for blocked packets by a factor of two.

2. New 2/4-cycle converters. The fixed width load pulse produced by the new FIFOs allowed the construction of a simplified, and much faster, 2/4-cycle conversion circuit for an interface to the external 2-cycle request/acknowledge signaling. This conversion circuit also introduces a limitation on the minimum cycle time for the output of a channel, which we must balance with an internal delay on the output request driving logic.

3. An improved decrementer. The old decrementer had badly sized transistors which resulted in very poor performance for decrementing large numbers.

4. Improved topology to minimize the length and capacitance of connecting wires, as well as to eliminate the need for any wasteful "padding" space previously needed to compose all the cells. As a result, the new MRC core is about 20% smaller.

SPICE simulations showed that the new MRC should indeed have much better performance than the original. To get a solid test of the new FIFO and 2/4-cycle converter stages, a 64-stage FIFO was constructed and sent out for fabrication in 3μm SCMOS at the end of January.

This FIFO returned early in April, and was promptly tested. The new FIFO fallthrough time is about 7.7ns, an improvement (same technology) of a factor of two over the original MRC. The Request$\rightarrow$Acknowledge cycle time is about 10ns, giving an overall cycle time of about 20ns (50M flits/s). This is five times faster than the original MRC, and exceeded our expectations by a factor of two! In a complete MRC, rather than just a simple FIFO, there will be longer wires and larger loads, but many circuits have also been tweaked slightly, so it should also have a 20ns cycle time. Fabrication in a 1.6μm feature size usually triples the speed of circuits, but in this case the cycle time will clearly be limited by the inductance of the chip leads. A low-inductance package will be critical for realizing the exceptional performance that the chip itself can deliver. We are expecting these designs to achieve a throughput well in excess of 100M flits/s.

All of the cells have now been laid out and individually simulated for the new MRC. A few simulations of compositions have to be performed next to try to minimize the internal cycle time. Then the complete MRC can be composed and switch-level simulated using Mossim and the AutoMossim driver program. It is expected that this can be done by late April and, barring unexpected problems, we should be able to send it to fabrication by early May.

If successful, we expect this new MRC chip to replace the MRC currently used in the Ametek Series 2010 multicomputer. With help from George Lewicki, this

design will also be transferred to an Intel fabrication process for use in a future Intel multicomputer.

4.6 Adaptive Routing in Multicomputer Networks

John Y. Ngai, Chuck Seitz

We continue to investigate the use of adaptive routing techniques to improve and sustain the performance of multicomputer communication networks. We have found what we believe is a scheme that is simple enough to be realizable in practice, and that outperforms even the highly evolved oblivious wormhole routing schemes. Completion of this work and publication of Ngai's thesis is expected in the next six months.

Our efforts have been divided into three different areas relating to three different aspects of the Adaptive Cut Through (ACT) routing technique:

(1) *Performance Analysis and Simulations:* Extensive simulations of various traffic patterns have been conducted. Some of the preliminary results were summarized in the last semiannual report. A detailed summary will appear in the dissertation.

(2) *Trial Implementation:* Here efforts are focused in isolating and understanding the major design trade-offs involved in a practical implementation of the ACT router. The investigation is conducted as a student group design project in the VLSI design class, with crucial contributions also from Charles Flaig and Glenn Lewis.

(3) *Reliability Enhancement Studies.* The single most important aspect of the routing formulation is its capability to exploit the existence of multiple paths intrinsic in most of the richly connected multicomputer networks. In addition to potential performance improvements, here our efforts are to investigate and evaluate the potential reliability enhancements that can be achieved. In particular, motivated by the desire to build high-performance networks through hardware realization of the routing operations, we look for the solution which allows us to continue the use of the original hardware routers systematically with little or no change in the routing hardware. To this end, we have developed a simple framework based on convexity and reachability defined with respect to the original routing relations. Extensive computations and simulations are conducted, with the result that the loss of a few percent of the routers or nodes will still allow well in excess of 80% of a multicomputer to remain in service.

4.7 The Notorious CIF-flogger Program

Glenn Lewis, Chuck Seitz

The CIF-flogger is a multicomputer program for flattening CIF files, rasterizing the geometry, and for performing parallel operations on the geometry in stripes. It runs under the CE/RK system, and hence, on most available multicomputers, including the Ametek Series 2010.

CIF-flogger currently supports simple bloat, shrink, and logical operations on the flattened geometry, and hence can perform most geometrical design-rule checks. It will eventually provide complete design-rule checking, well checks, and circuit extraction. Based on timings on the iPSC/1, CIF-flogger is expected to perform design rule checks for 100K-transistor chips in much less than 1s per rule on second-generation multicomputers.

4.8 Pads and Pad Frame Generation

Charles Flaig, Glenn Lewis, Chuck Seitz

Motivated in large part by the variety of mesh routing chips (MRCs) being designed, a similarly large variety of new pad circuits have been designed for $\lambda = 0.6\mu$m, 0.8μm, and 1.0μm MOSIS SCMOS processes. The unusual features of these designs include:

1. The use of longitudinal (bipolar) clamp transistors for static and overvoltage protection. These protection circuits appear to be very effective.

2. Experimental use of pad spacings that are less than the standard MOSIS 200μm pad pitch. MRCs run with $\lambda = 0.8\mu$m have used a $191\lambda = 152.8\mu$m= 6.02mil pad pitch with 33 pads per edge. When one run of 50 chips was bonded in the standard MOSIS 132-pin PGA package* (small package well variety), we observed 83% yield on this MRC overall, and 100% bonding yield. Output edge times were less than 2ns, and these (self-timed) MRCs operate at about 30Mflits/s.

Our thanks to George Lewicki at MOSIS for tolerating and perhaps even encouraging these experiments.

These efforts, and related efforts in helping MOSIS with standard frames, have required the generation of many pad frames. Thus, the pad library was created along with some tools that have automated generation of pad frames, and have saved countless hours of tedious work.

* Other users of the MOSIS 132PGA packages are advised to study the documentation on this very nice (as PGAs go) package, noting in particular that 12 of the pins have about 5× lower resistance and inductance than the rest. We have used these pins for Vdd and GND.

4.9 SunCIFP

Glenn Lewis, Wen-King Su, Chuck Seitz

A new version of the CIFP program has been written and is available for distribution. It runs on Sun workstations, and creates a display of CIF geometry on a Sun window.

To be published in the Proceedings of the 1988 Hypercube Conference

The Architecture and Programming of the Ametek Series 2010 Multicomputer

Charles L. Seitz, William C. Athas, Charles M. Flaig,
Alain J. Martin, Jakov Seizovic, Craig S. Steele, Wen-King Su
Department of Computer Science
California Institute of Technology

Background

During the period following the completion of the Cosmic Cube experiment [1], and while commercial descendants of this first-generation multicomputer (message-passing concurrent computer) were spreading through a community that includes many of the attendees of this conference, members of our research group were developing a set of ideas about the physical design and programming for the second generation of medium-grain multicomputers.

Our principal goal was to improve by as much as two orders of magnitude the *relationship* between message-passing and computing performance, and also to make the topology of the message-passing network practically invisible. Decreasing the communication latency relative to instruction execution times extends the application span of multicomputers from easily partitioned and distributed problems (*eg*, matrix computations, PDE solvers, finite element analysis, finite difference methods, distant or local field many-body problems, FFTs, ray tracing, distributed simulation of systems composed of loosely coupled physical processes) to computing problems characterized by "high flux" [2] or relatively fine-grain concurrent formulations [3, 4] (*eg*, searching, sorting, concurrent data structures, graph problems, signal processing, image processing, and distributed simulation of systems composed of many tightly coupled physical processes). Such applications place heavy demands on the message-passing network for high bandwidth, low latency, and non-local communication. Decreased message latency also improves the efficiency of the class of applications that have been developed on first-generation systems, and the insensitivity of message latency to process placement simplifies the concurrent formulation of application programs.

Our other goals included a streamlined and easily layered set of message primitives, a node operating system based on a reactive programming model, open interfaces for accelerators and peripheral devices, and node performance improvements that could be achieved economically by using the same technology employed in contemporary workstation computers.

By the autumn of 1986, these ideas had become sufficiently developed, molded together, and tested through simulation to be regarded as a complete architectural design. We were fortunate that the Ametek Computer Research Division was ready and willing to work with us to develop this system as a commercial product. The Ametek Series 2010 multicomputer is the result of this joint effort.

Architecture

Overview

Each Ametek Series 2010 node includes a 25MHz Motorola 68020 processor with a M68881 or M68882 floating-point coprocessor, zero-wait-state memory management hardware, up to 8MB of memory, and a VME interface for accelerators or peripheral controllers. These nodes are about an order of magnitude faster and have about an order of magnitude more memory than those in the first-generation systems. The multicomputer is normally hosted from Sun-3 workstation computers, which also use M68020 processors; hence, the native Sun compilers are able to generate process code for the nodes.

What most distinguishes the Ametek Series 2010 multicomputer from the first-generation "hypercubes" is its message-routing and message-handling hardware. Given our objective not only of keeping pace with the order-of-magnitude advance in node computing performance, but of improving the relationship between communication and computing latencies, we were seeking a major improvement in communication performance.

The way in which this improvement in message performance was achieved was with a combination of organization and technology. The Ametek 2010 does not use a binary n-cube (hypercube) connection network, but instead uses a two-dimensional routing mesh of high-performance custom routing chips. This low-dimension network minimizes latency for a given wire bisection of

the network by allowing more parallel wires and higher bandwidth for each channel. The "wormhole" routing method, unlike store-and-forward routing, does not use storage bandwidth or computing cycles in nodes through which a message is routed. Packets are injected into the network by the source node and leave the network only at the destination node. The entire edge of the mesh is available for hosts or peripheral devices. In order to reduce the software component of the message latency, the nodes include a microprogrammed second processor that manages the send and receive queues.

Communication Network

The Ametek Series 2010 message network is composed of a two-dimensional mesh of custom Mesh Routing Chips (MRCs) [5]. The communication channels are 8 bits wide, and operate self-timed at well in excess of 20MHz, yielding a communication bandwidth per channel of at least 20MB/s (160Mb/s). A higher channel bandwidth is feasible but not economic, since it would exceed even the sequential-access memory bandwidth in the nodes. A node that is sending and receiving concurrently at 20MB/s must on average be performing ten 32-bit accesses per μs.

Message packets advance directly from MRC to MRC in a blocking variant of cut-through routing [6] that we call "wormhole" routing [3,5,7]. The time required to advance the head of a packet from MRC to MRC is only about two byte times. Thus, for example, the time required to send a 64-byte packet (8 double-precision floating-point operands) from corner to corner in a 64-node 8×8 mesh (distance 14) is $0.05(2\times14+64)\mu s = 4.6\mu s$. One may think of this packet as requiring 1.4μs for path formation and an additional 3.2μs to spool the message through the channels. For message lengths that are typical of medium-grain multicomputer programs, the length in bytes is considerably larger than the distance in the mesh; hence, the length-dependent component of the latency dominates, and the latency exhibits little sensitivity to message distance.

The performance of this wormhole routing network cannot be compared by a single measure with the performance of the software-controlled store-and-forward packet cut-through message systems in first-generation multicomputers. The store-and-forward networks consume storage bandwidth and computing cycles in the routing nodes, while accumulating a latency of several hundred μs per hop. The case that is most critical for exploiting finer-grain concurrency (*eg*, relatively fewer instructions between message operations, and typically shorter messages) is short non-local messages. The same corner-to-corner message that is delivered in 4.6μs by the Ametek Series 2010 message network would be handled in a store-and-forward binary 6-cube by the source, destination, and five intermediate nodes, with a total latency of several ms. Thus, in the important case of relatively short non-local messages, the reduction in message latency approaches three orders of magnitude.

The scaling and congestion properties of the network require some comment. In conditions of large applied load to a mesh network, the performance is largely determined by the bisection. Hence, it is desirable to keep the mesh configurations as close to square as possible. A 4×16 64-node machine will function correctly, but has a smaller bisection than an 8×8 configuration. Under an assumption of fixed wire bisection, a two-dimensional network minimizes latency [3,8] for our centerline design point of $N = 256$, a 16×16 mesh. Smaller machines have a surplus of network bandwidth, while larger machines are capable with intense, non-localized message traffic of driving the message network to a state of moderate congestion and consequent noticeable latency. However, according to our simulations, low-dimension networks are very effective in source-queueing packets when the applied load exceeds the network capacity, such that the throughput of the network remains close to its peak operating point.

To realize this scaling in practice, the basic packaging unit in the Ametek Series 2010 is a 4×4 submesh of 16 nodes. The 4×4 submesh is built as an active backplane, measuring 17×12 inches^2, into which the node boards are plugged. These submeshes can be connected vertically and horizontally with other 4×4 submesh units to construct systems up to $32\times16 = 512$ nodes. Still larger systems are perfectly feasible; however, to confine their vertical dimension, they would be constructed with special backplanes with 2×8 or 1×16 submesh units.

Node Architecture

The small network component of the message latency, although important in part for avoiding congestion by sending packets through the network in short bursts, requires equal attention to minimizing the "startup" time or software component of the message latency. The message primitives have accordingly been streamlined so that messages are sent and received from dynamically allocated memory, and the node is an unsymmetrical two-processor architecture. The M68020 and a microprogram-controlled message interface processor share access to main memory and cooperatively maintain data structures consisting of linked control blocks that point to message pages. One structure includes the receive queue and preallocated pages for incoming messages, and the other includes the send queue. Block transfers between memory and hardware queues in the message interface processor are accomplished in static column mode, one of the efficient, high-bandwidth, sequential-access modes of modern dRAM chips. The main memory bandwidth in this mode is a 32-bit cycle each 80ns, or 50MB/s.

Static column mode is also used for the M68020 access, with the most recently accessed column in each 1MB bank serving as a 2KB fast page, similar in effect to a cache set. Thus, a typical 4MB node maintains four fast pages from which the 25MHz M68020 can run with no wait states. Nodes with more memory have proportionately more fast pages. The dRAM refresh is accomplished by hardware. The address translation unit is implemented

with fast static RAMs, with 8KB pages for the code, data, and stack regions, and 256B pages for the dynamically allocated message region. Regions associated with the same process are normally mapped into separate banks so that contiguous code, data, stack, and message references will introduce no wait states.

Messages that are longer than 256B are fragmented into packets with 256B payloads, so that long messages will not block other traffic in the message system for long periods. The size of message pages and the maximum packet length are the same, so that fragmentation and reassembly are accomplished without copying. Taken together, the use of a fast dRAM sequential access mode and the remapping of packets to messages is very effective. Even with the software overhead of fragmenting and reassembling long messages, the asymptotic bandwidth in sending long messages from node to node is higher than the bandwidth that the M68020 achieves copying blocks within the memory of a single node.

The Ametek Series 2010 node design does not compromise in any way with protection. A user process can access only its own data and messages. The node hardware is designed to support not only multiprogramming, but multiple users and virtual memory operation.

Each node also has a high-performance VME interface for peripheral controllers (such as disk interfaces) and accelerators (such as a standard 20Mflop floating-point vector processor).

Programming

The Ametek Series 2010 employs the same process model that was supported on the Cosmic Cubes and first-generation commercial systems. However, in order to streamline the message handling and to allow for efficient layering of a variety of message functions, the primitive message functions are quite different from those used in the first-generation multicomputers. The programming system described here [9] was developed in our research group, and has been in regular use for the past year on the Cosmic Cubes and other multicomputers operated by our group. It was ported to the Ametek Series 2010 without any notable difficulties.

Node Operating System

The standard node operating system for the Ametek 2010 is a proprietary adaptation of a new multicomputer operating system, the *Reactive Kernel* (RK). RK is based on a small kernel that dispatches to kernel processes called *handlers* according to the *tag* in the message at the head of the receive queue. Different handlers and their associated user library routines support different sets of message primitives, as may be required for different languages and applications. Different handlers may be coresident in "subcubes" of the Ametek 2010, so that in the usual *space sharing* mode of operation, different programs can be run concurrently with different primitives. With a suitable handler and library, the Ametek Series 2010 can support the message primitives of any of the first-generation multicomputers.

Host Runtime System

The host runtime system is derived from the Cosmic Environment system (CE version 7.2). The CE system consists of a set of daemon processes, utility programs, and library routines. It handles the allocation of one or more multicomputers, and supports uniform communication between UNIX and node processes. The UNIX processes may all be on the hardware host, or may be distributed among multiple hosts on the same network.

The CE system supports numerous program development features, and is commonly run not only as a host runtime system for multicomputers, but also on single UNIX systems, across networks of UNIX systems, and on multiprocessors. It is a "combat-proven" system that has now been distributed to more than 100 sites. Instructions for obtaining a CE distribution are included in the programming guide [9], which is available from the Caltech Computer Science librarian.

User Programming

The *reactive handler* and its associated library support a set of user interface routines that are analogous to those used in the system interface between a hander and the kernel. Although illustrated here as they are called from processes written in C, user interface routines also exist for other languages.

As usual, each process has a unique identifier consisting of the node number and a process identifier within the node, *viz:* (`node, pid`). Process spawning is dynamic, and can be initiated from any node or host process with the function:

```
spawn("filename", node, pid, "")
```

Also, as usual, messages are directed to processes, and are queued in transit, but message order is preserved between pairs of communicating processes. Within the limits of the computation being deterministic and not exceeding available storage sizes, the results of a computation do not depend on the way in which the processes are distributed.

Messages are sent and received from dynamically allocated memory that is accessed both by user processes and by the message system. Message buffers are arrays of bytes with no presumed structure, and the C functions that return pointers to message buffers return maximally aligned pointers of type `char*`. Message space can be allocated by:

```
p = xmalloc(length);
```

where the `length` of the block pointed to by p is specified in bytes, and can be deallocated by:

```
xfree(p);
```

These functions are semantically identical to the usual UNIX `malloc` and `free` functions.

When a message has been built in a block that has been allocated, its contents can be sent as a message by:

```
xsend(p, node, pid);
```

The xsend function also deallocates the message block; that is, xsend(p,...) is like xfree(p), except that it also sends a message. Thus, there is no need for blocking or for feedback that the message has been sent. When the function returns, the message block is gone.

Messages can be received by:

```
p = xrecvb();
```

such that p then points to a new message. As indicated by the "b" at the end of xrecvb, this is a *blocking* function that does not return until a message has arrived for the process.

The execution of the xrecvb function is just like allocating a message buffer with xmalloc, except that the length of the block allocated is determined by the length of the message received. Once the message contents are no longer needed, the allocated space should be freed. Of course, the message space can be freed with xfree(p), but it can also be freed by xsend(p, ...) if there is a message of the same length to send. It frequently happens in message-passing programs that a message that is received is simply modified by a computation and then sent on to another process.

The non-blocking receive function is called xrecv. It is required only for applications in which a process may need to *probe* for another message without giving up the right to continue execution. The usage of the xrecv function is identical to xrecvb; however, it may return a NULL pointer if there is no message queued for this process. This behavior of the xrecv function allows one to write programs that can do other work while waiting for a message; for example:

```
while (1) {
    if (p = xrecv()) digest(p);
    else do_other_work();
}
```

In such usage, the digest(p) and do_other_work() functions should return after a bounded time to call xrecv again, because calling xrecv or xrecvb when the next message in the node's receive queue is for another process allows the kernel to save the state of that process and start running the other process. The appearance of xrecv or xrecvb in the code marks a *choice point* for switching the execution to another process, and it is in this sense that the scheduling is *reactive* or *message driven*.

Programs may use these primitive functions directly, or may use other classes of functions that are expressed in terms of the "x" primitives. Extra information, such as a message type, can be inserted into extra space allocated in a message buffer, and sent with a message. The function used to receive typed messages can filter them into separate queues of pointers (the messages themselves remain intact) according to the type. For example, the user interface functions defined for FORTRAN allow processes to exercise discretion in the messages received according to any combination of message type and sender ID.

Conclusion

Taken together, the computing and communication performance, scalability, open interfaces, I/O capability, new features, and system software of this second-generation multicomputer represent to us the fulfillment of an "IOU" — a working demonstration of the capabilities we have said would be possible to include in a well-engineered multicomputer.

Acknowledgments

The research that led to the architectural design and system software of the Ametek Series 2010 was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by a grant from Ametek Computer Research Division.

We very much appreciate the dedicated efforts and support of the employees and management of Ametek.

Certain of the techniques described here are the subjects of patents filed by Caltech and by Ametek. The Cosmic Environment and Reactive Kernel are the property of Caltech, and are licensed to Ametek.

References

[1] Charles L Seitz, "The Cosmic Cube," *CACM*, 28(1): 22-33, January 1985.

[2] J D Ullman, "Flux, Sorting, and Supercomputer Organization for AI Applications," *J of Parallel and Distributed Computing* 1: 133-151, 1984.

[3] William J Dally, *A VLSI Architecture for Concurrent Data Structures*, Kluwer Academic Publishers, 1987.

[4] William C Athas, "Fine Grain Concurrent Computations," Caltech Computer Science technical report (PhD thesis) 5242:TR:87.

[5] Charles M Flaig, "VLSI Mesh Routing Systems," Caltech Computer Science technical report (MS thesis) 5241:TR:87.

[6] P Kermani and L Kleinrock, "Virtual Cut-through: A New Computer Communication Switching Technique," *Computer Networks* 3: 267-286, 1979.

[7] William J Dally, Charles L Seitz, "The Torus Routing Chip," *Distributed Computing* 1(4): 187-196, Springer International, 1986.

[8] William J Dally, "Wire-Efficient VLSI Multiprocessor Communication Networks," *Proc 1987 Stanford Conference on Advanced Research in VLSI*, MIT Press, 1987.

[9] Charles L Seitz, Jakov Seizovic, Wen-King Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming," Caltech-CS-TR-88-1, 1988.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports
April, 1988
Available from the Computer Science Department Library
Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

If you wish to order any of the reports listed, complete this form and return it with your check or international money order (in U.S. dollars) payable to CALTECH. **Prepayment is required for all materials.**

____CS-TR-88-01 $3.00 *C Programmer's Abbreviated Guide to Multicomputer Programming*,
Seitz, Charles, Jakov Seizovic and Wen-King Su

____5258:TR:88 $3.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5256:TR:87 $2.00 *Synthesis Method for Self-timed VLSI Circuits*,
Martin, Alain
current supply only: see *Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design*, 224-229, Oct'87

____5251:TR:87 $2.00 *Conditional Knowledge as a Basis for Distributed Simulation*,
Chandy, K. Mani and Jay Misra

____5250:TR:87 $10.00 *Images, Numerical Analysis of Singularities and Shock Filters*, PhD Thesis
Rudin, Leonid Iakov

____5249:TR:87 $6.00 *Logic from Programming Language Semantics*, PhD Thesis
Choo, Young-il

____5247:TR:87 $6.00 *VLSI Concurrent Computation for Music Synthesis*, PhD Thesis
Wawrzynek, John

____5246:TR:87 $3.00 *Framework for Adaptive Routing*
Ngai, John Y and Charles L. Seitz

____5244:TR:87 $3.00 *Multicomputers*
Athas, William C and Charles L Seitz

____5243:TR:87 $5.00 *Resource-Bounded Category and Measure in Exponential Complexity Classes*, PhD Thesis
Lutz, Jack H

____5242:TR:87 $8.00 *Fine Grain Concurrent Computations*, PhD Thesis
Athas, William C.

____5241:TR:87 $3.00 *VLSI Mesh Routing Systems*, MS Thesis
Flaig, Charles M

____5240:TR:87 $2.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5239:TR:87 $3.00 *Trace Theory and Systolic Computations*
Rem, Martin

____5238:TR:87 $7.00 *Incorporating Time in the New World of Computing System*, MS Thesis
Poh, Hean Lee

____5236:TR:86 $4.00 *Approach to Concurrent Semantics Using Complete Traces*, MS Thesis
Van Horn, Kevin S.

____5235:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5234:TR:86 $3.00 *High Performance Implementation of Prolog*
Newton, Michael O

____5233:TR:86 $3.00 *Some Results on Kolmogorov-Chaitin Complexity*, MS Thesis
Schweizer, David Lawrence

____5232:TR:86 $4.00 *Cantor User Report*
Athas, W.C. and C. L. Seitz

_5231:TR:86 $2.00 *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*
Dally, William J and Charles L Seitz
current supply only: see *IEEE Transactions on Computers* vol C-36 no 5, May 1987

_5230:TR:86 $24.00 *Monte Carlo Methods for 2-D Compaction*, PhD Thesis
Mosteller, R.C.

_5229:TR:86 $4.00 *anaLOG - A Functional Simulator for VLSI Neural Systems*, MS Thesis
Lazzaro, John

_5228:TR:86 $3.00 *On Performance of k-ary n-cube Interconection Networks*,
Dally, Wm. J

_5227:TR:86 $18.00 *Parallel Execution Model for Logic Programming*, PhD Thesis
Li, Pey-yun Peggy

_5223:TR:86 $15.00 *Integrated Optical Motion Detection*, PhD Thesis
Tanner, John E.

_5221:TR:86 $3.00 *Sync Model: A Parallel Execution Method for Logic Programming*
Li, Pey-yun Peggy and Alain J. Martin
current supply only: see *Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86*

_5220:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

_5215:TR:86 $2.00 *How to Get a Large Natural Language System into a Personal Computer*,
Thompson, Bozena H. and Frederick B. Thompson

_5214:TR:86 $2.00 *ASK is Transportable in Half a Dozen Ways*,
Thompson, Bozena H. and Frederick B. Thompson

_5212:TR:86 $2.00 *On Seitz' Arbiter*,
Martin, Alain J

_5210:TR:86 $2.00 *Compiling Communicating Processes into Delay-Insensitive VLSI Circuits*,
Martin, Alain
current supply only: see *Distributed Computing* v 1 no 4 (1986)

_5209:TR:86 $11.00 *VLSI Architecture for Concurrent Data Structures*, PhD Thesis,
Dally, William J.
current supply only: see book published by Kluwer, 1987

_5208:TR:86 $2.00 *The Torus Routing Chip*,
Dally, William and Charles L Seitz
current supply only: see *Distr. Computing* vol 1 no 4 1986

_5207:TR:86 $2.00 *Complete and Infinite Traces: A Descriptive Model of Computing Agents*,
van Horn, Kevin

_5205:TR:85 $2.00 *Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity*,
Schweizer, David and Yaser Abu-Mostafa

_5204:TR:85 $3.00 *An Inverse Limit Construction of a Domain of Infinite Lists*,
Choo, Young-Il

_5202:TR:85 $15.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

_5200:TR:85 $18.00 *ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation*, PhD thesis
Whelan, Dan

_5198:TR:85 $8.00 *Neural Networks, Pattern Recognition and Fingerprint Hallucination*, PhD thesis
Mjolsness, Eric

_5197:TR:85 $7.00 *Sequential Threshold Circuits*, MS thesis
Platt, John

_5195:TR:85 $3.00 *New Generalization of Dekker's Algorithm for Mutual Exclusion*,
Martin, Alain J
current supply only: see *Information Processing Letters*, **23**, 295-297 (1986)

_5194:TR:85 $5.00 *Sneptree - A Versatile Interconnection Network*,
Li, Pey-yun Peggy and Alain J Martin

___5193:TR:85 $2.00 *Delay-insensitive Fair Arbiter*
Martin, Alain J
current supply only: see *Distr Computing* **1**:226-234 (1986)

___5190:TR:85 $3.00 *Concurrency Algebra and Petri Nets*,
Choo, Young-il

___5189:TR:85 $10.00 *Hierarchical Composition of VLSI Circuits*, PhD Thesis
Whitney, Telle

___5185:TR:85 $11.00 *Combining Computation with Geometry*, PhD Thesis
Lien, Sheue-Ling

___5184:TR:85 $7.00 *Placement of Communicating Processes on Multiprocessor Networks*, MS Thesis
Steele, Craig

___5179:TR:85 $3.00 *Sampling Deformed, Intersecting Surfaces with Quadtrees*, MS Thesis,
Von Herzen, Brian P.

___5178:TR:85 $9.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

___5177:TR:85 $4.00 *Hot-Clock nMOS*,
Seitz, Charles, A H Frey, S Mattisson, S D Rabin, D A Speck, and J L A van de Snepscheut
current supply only: see *Proc 1985 Chapel Hill Conference on VLSI*, p 1-17

___5174:TR:85 $7.00 *Balanced Cube: A Concurrent Data Structure*,
Dally, William J and Charles L Seitz

___5172:TR:85 $6.00 *Combined Logical and Functional Programming Language*,
Newton, Michael

___5168:TR:84 $3.00 *Object Oriented Architecture*,
Dally, Bill and Jim Kajiya

___5165:TR:84 $4.00 *Customizing One's Own Interface Using English as Primary Language*,
Thompson, B H and Frederick B Thompson

___5164:TR:84 $13.00 *ASK French - A French Natural Language Syntax*, MS Thesis
Sanouillet, Remy

___5160:TR:84 $7.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

___5158:TR:84 $6.00 *VLSI Architecture for Sound Synthesis*,
Wawrzynek, John and Carver Mead

___5157:TR:84 $15.00 *Bit-Serial Reed-Solomon Decoders in VLSI*, PhD Thesis
Whiting, Douglas

___5148:TR:84 $4.00 *Fair Mutual Exclusion with Unfair P and V Operations*,
Martin, Alain and Jerry Burch
current supply only: see *Information Processing Letters*, **21**, 97-100, (1985)

___5147:TR:84 $4.00 *Networks of Machines for Distributed Recursive Computations*,
Martin, Alain and Jan van de Snepscheut

___5143:TR:84 $5.00 *General Interconnect Problem*, MS Thesis
Ngai, John

___5140:TR:84 $5.00 *Hierarchy of Graph Isomorphism Testing*, MS Thesis
Chen, Wen-Chi

___5139:TR:84 $4.00 *HEX: A Hierarchical Circuit Extractor*, MS Thesis
Oyang, Yen-Jen

___5137:TR:84 $7.00 *Dialogue Designing Dialogue System*, PhD Thesis
Ho, Tai-Ping

___5136:TR:84 $5.00 *Heterogeneous Data Base Access*, PhD Thesis
Papachristidis, Alex

___5135:TR:84 $7.00 *Toward Concurrent Arithmetic*, MS Thesis
Chiang, Chao-Lin

___5134:TR:84 $2.00 *Using Logic Programming for Compiling APL*, MS Thesis
Derby, Howard

5133:TR:84 $13.00 *Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems*, PhD Thesis
Lin, Tzu-mu

5132:TR:84 $10.00 *Switch Level Fault Simulation of MOS Digital Circuits*, MS Thesis
Schuster, Mike

5129:TR:84 $5.00 *Design of the MOSAIC Processor*, MS Thesis
Lutz, Chris

5128:TM:84 $3.00 *Linguistic Analysis of Natural Language Communication with Computers*,
Thompson, Bozena H

5125:TR:84 $6.00 *Supermesh*, MS Thesis
Su, Wen-king

5124:TR:84 $4.00 *Probe: An Addition to Communication Primitives*,
Martin, Alain
current supply only: see *Information Processing Letters*, **20**, no 3, (1985)

5123:TR:84 $14.00 *Mossim Simulation Engine Architecture and Design*,
Dally, Bill

5122:TR:84 $8.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

5114:TM:84 $3.00 *ASK As Window to the World*,
Thompson, Bozena, and Fred Thompson

5112:TR:83 $22.00 *Parallel Machines for Computer Graphics*, PhD Thesis
Ulner, Michael

5106:TM:83 $1.00 *Ray Tracing Parametric Patches*,
Kajiya, James T

5104:TR:83 $9.00 *Graph Model and the Embedding of MOS Circuits*, MS Thesis
Ng, Tak-Kwong

5097:TR:83 $4.00 *Design of a Self-timed Circuit for Distributed Mutual Exclusion*,
Martin, Alain J
current supply only: see *Proc. Chapel Hill Conf. on VLSI*, 245-259, May 1985

5094:TR:83 $2.00 *Stochastic Estimation of Channel Routing Track Demand*,
Ngai, John

5092:TM:83 $2.00 *Residue Arithmetic and VLSI*,
Chiang, Chao-Lin and Lennart Johnsson

5091:TR:83 $2.00 *Race Detection in MOS Circuits by Ternary Simulation*,
Bryant, Randal E

5090:TR:83 $9.00 *Space-Time Algorithms: Semantics and Methodology*, PhD Thesis
Chen, Marina Chien-mei

5089:TR:83 $10.00 *Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits*,
Lin, Tzu-Mu and Carver A Mead

5086:TR:83 $4.00 *VLSI Combinator Reduction Engine*, MS Thesis
Athas, William C Jr

5082:TR:83 $10.00 *Hardware Support for Advanced Data Management Systems*, PhD Thesis
Neches, Philip

5081:TR:83 $4.00 *RTsim - A Register Transfer Simulator*, MS Thesis
Lam, Jimmy

5080:TR:83 $4.00 *Distributed Mutual Exclusion on a Ring of Processes*,
Martin, Alain
current supply only: see *Science of Computer Programming*, **5**, (1985)

5079:TR:83 $2.00 *Highly Concurrent Algorithms for Solving Linear Systems of Equations*,
Johnsson, Lennart
current supply only: see *Acta Informatica* **20**, 301-313, (1983)

5074:TR:83 $10.00 *Robust Sentence Analysis and Habitability*,
Trawick, David

___5073:TR:83 $12.00 *Automated Performance Optimization of Custom Integrated Circuits*, PhD Thesis
Trimberger, Steve

___5065:TR:82 $3.00 *Switch Level Model and Simulator for MOS Digital Systems*,
Bryant, Randal E

___5054:TM:82 $3.00 *Introducing ASK, A Simple Knowledgeable System*, Conf on App'l Natural Language Processing
Thompson, Bozena H and Frederick B Thompson

___5051:TM:82 $2.00 *Knowledgeable Contexts for User Interaction*, Proc Nat'l Computer Conference
Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho

___5035:TR:82 $9.00 *Type Inference in a Declarationless, Object-Oriented Language*, MS Thesis
Holstege, Eric

___5034:TR:82 $12.00 *Hybrid Processing*, PhD Thesis
Carroll, Chris

___5033:TR:82 $4.00 *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*,
Schuster, Mike, Randal Bryant and Doug Whiting

___5029:TM:82 $4.00 *POOH User's Manual*,
Whitney, Telle

___5018:TM:82 $2.00 *Filtering High Quality Text for Display on Raster Scan Devices*,
Kajiya, Jim and Mike Ullner

___5017:TM:82 $2.00 *Ray Tracing Parametric Patches*,
Kajiya, Jim

___5015:TR:82 $15.00 *VLSI Computational Structures Applied to Fingerprint Image Analysis*,
Megdal, Barry

___5014:TR:82 $15.00 *Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, PhD Thesis
Lang, Charles R Jr

___5012:TM:82 $2.00 *Switch-Level Modeling of MOS Digital Circuits*,
Bryant, Randal

___5000:TR:82 $6.00 *Self-Timed Chip Set for Multiprocessor Communication*, MS Thesis
Whiting, Douglas

___4684:TR:82 $3.00 *Characterization of Deadlock Free Resource Contentions*,
Chen, Marina, Martin Rem, and Ronald Graham

___4655:TR:81 $20.00 *Proc Second Caltech Conf on VLSI*,
Seitz, Charles, ed.

___3760:TR:80 $10.00 *Tree Machine: A Highly Concurrent Computing Environment*, PhD Thesis
Browning, Sally

___3759:TR:80 $10.00 *Homogeneous Machine*, PhD Thesis
Locanthi, Bart

___3710:TR:80 $10.00 *Understanding Hierarchical Design*, PhD Thesis
Rowson, James

___3340:TR:79 $26.00 *Proc. Caltech Conference on VLSI (1979)*,
Seitz, Charles, ed

___2276:TM:78 $12.00 *Language Processor and a Sample Language*,
Ayres, Ron

Caltech Computer Science Technical Reports

Please fill in your name, address and amount enclosed below:

Name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125